Design of resilient systems

Architectural, paradigmatic and algorithmic issues

a Tutorial

Paulo Veríssimo
Univ. of Lisboa Faculty of Sciences
Lisboa – Portugal
pjv@di.fc.ul.pt
http://www.navigators.di.fc.ul.pt

Pointers to tutorial material

- **With the aim of disseminating resilience and intrusion tolerance concepts and techniques to a wide audience, the following documents are available from the University of Lisboa web site.**
- http://www.navigators.di.fc.ul.pt/it/index.htm

- Intrusion-Tolerant Architectures: Concepts and Design *(Extended version).* Veríssimo, P. E., and Neves, N. F., and Correia, M. P. In: Architecting Dependable Systems. Springer-Verlag LNCS 2677 (2003). Technical Report DI/FCUL TR03-5, Dept. of Informatics, University of Lisboa (2003). abstract – pdf
- Intrusion-tolerant middleware: The road to automatic security. P. Verissimo, N. F. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I.Welch. *IEEE Security & Privacy*, 4(4):54-62, Jul./Aug. 2006.
- Intrusion-Resilient Middleware Design and Validation. P. Verissimo, M. Correia, N. Neves, P. Sousa. “Annals of Emerging Research in Information Assurance, Security and Privacy Services”. Elsevier 2008 (to appear).

Why do systems need resilience?

- a non-canonical definition of resilience:
  - "Ability to recover from or adjust easily to misfortune or change."
- the case for resilience:
  1. we want systems to operate through faults and attacks in a seamless manner, in an automatic way
- intrusion tolerance lets us achieve that
  2. operating conditions and environments are everyday more uncertain and/or hostile
  3. we want to deploy systems in unattended manner
- intrusion tolerance insufficient
  4. we need extra predicates

Designing for resilience

- Architecting intrusion-tolerant systems
  - usual InTol systems live off some middleware layers that mask failures below, used by upper layers transparently of how tolerance is achieved
  - middleware is generally composed of *n* replicas cooperating through distributed protocols
- Tolerating Intrusions
  - replicas are attacked and corrupted at the measure of the power of threats (attacks, accidents)
  - as long as there are sufficient replicas to perform the service correctly, the system continues to function
  - … sometimes even without the user noticing anything

Designing for resilience

- Handling Attack Severity
  - expected threats are severe (e.g., malicious intelligence), so protocols should resist to arbitrary faults (i.e., Byzantine)
  - necessary quorum for Byzantine resilience to faults is typically $n = 3f + 1$ replicas
  - for InTol middleware, the goal is to always preserve the number of replicas above the minimum threshold mentioned above

Designing for resilience

- Resisting Attacks
  - faults and attacks erode systems inexorably so an unattended (automatic) system faces inevitable resource exhaustion which leads to inevitable failure
  - threats are so intense that this is not an academic possibility: they are exacerbated by attacker power and common-mode vulnerabilities
  - additional defences are often required to shrink attackers' chances and slow down the rate of failures in order to prevent resource exhaustion: diversity, obfuscation, hybridization, trusted-trustworthy components, rejuvenation

Designing for resilience

- Validating attacks
  - necessary to study and understand malicious faults in order to validate the fault assumptions underlying the above-mentioned intrusion-tolerant algorithms
  - for InTol middleware, this would allow algorithm and system designers to introduce more realistic assumptions
  - we are still far from a thorough understanding of the mechanisms behind the trilogy attack-vulnerability-intrusion

Further Reading

Further Reading

- E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, M. Herrb, Lessons learned from the deployment of a high-interaction honeypot, Proceedings of the 6th European Dependable Computing Conference, October 2006, Coimbra, Portugal , pp *39-46*
- L. Alvisi, D. Malkhi, E. Pierce, M. K. Reiter, and R. N. Wright, Dynamic Byzantine quorum systems," in Proc. Int'l Conference on Dependable Sys and Networks (FTCS-30/DCCA-8), pp. 283-292, 2000.
- Y. Amir et al. Secure group communication in asynchronous networks with failures: Integration and experiments. In Proc. The 20th IEEE International Conference on Distributed Computing Systems (ICDCS 2000), pages 330-343, Taipei, Taiwan, April 2000.
- Y. Amir, C. Danilov, J. Kirsch, J. Lane, D. Dolev, C. Nita-Rotaru, J. Olsen, and D. Zage. Scaling Byzantine fault-tolerant replication towide area networks. In Proceedings of the International Conference on Dependable Systems and Networks, pages 105–114, June 2006.
- G. Ateniese, M. Steiner, G. Tsudik: Authenticated group key agreement and friends. In Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS-98), pages 17-26, New York, November 3-5 1998. ACM Press.
- Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multi-party authentication services and key agreement protocols. IEEE Journal of Selected Areas on Communications, 18, March 2000.
- A. N. Bessani, M. Correia, J. S. Fraga, and L. C. Lung. Decoupled quorum-based Byzantine-resilient coordination in open distributed systems. In Proceedings of the 6th IEEE International Symposium on Network Computing and Applications, pages 231–238, July 2007.
- Christian Cachin. Distributing Trust on the Internet. In Procs. of the Int'l Conf. on Depend. Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.
- C. Cachin, K. Kursawe and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography", in Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC), pp.123-32, 2000b.

Design for Resilience

Further Reading

- C. Cachin and J. A. Poritz, Hydra: Secure replication on the Internet," In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- M. Castro and B. Liskov, Practical Byzantine fault tolerance," in Proc. Third Symp. Operating Systems Design and Implementation (OSDI), 1999.
- M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 20(4):398–461, Nov. 2002.
- M. Castro, R. Rodrigues, and B. Liskov. BASE: Using abstraction to improve fault tolerance. ACM Transactions Computer Systems, 21(3):236–269, Aug. 2003.
- T. D. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems," Journal of the ACM, vol. 43, no. 2, pp. 225-267, 1996.
- Nick Cook, Santosh Shrivastava, Stuart Wheater. Distributed Object Middleware to Support Dependable Information Sharing between Organisations. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- M. Correia, N.F. Neves, P. Veríssimo, Lau Cheuk Lung, Low Complexity Byzantine-Resilient Consensus, Distributed Computing, vol. 17, n. 3, pp. 237--249, March 2005.
- M. Correia, N. F. Neves, and P. Ver´ıssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. Computer Journal, 41(1):82–96, Jan. 2006.
- M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems, pages 74–183, Oct. 2004.
- M. Correia, Lau Cheuk Lung, Nuno Ferreira Neves, and P. Veríssimo. Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model. In Proc. of Symp. of Reliable Distributed Systems, October 2002, Japan.
- M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In proceedings of the EDCC-4, Fourth European Dependable Computing Conference, Toulouse, France - October 23-25, 2002.

Further Reading

- W. S. Dantas, A. N. Bessani, J. Fraga, and M. Correia. Evaluating Byzantine quorum systems. In Proceedings of the 26th IEEE Symposium on Reliable Distributed Systems, Oct. 2007.
- H. Debar, M. Dacier, A. Wespi: Towards a taxonomy of intrusion detection systems. Computer Networks, 31:805-822, 1999.
- Y. Desmedt: Society and group oriented cryptography: a new concept; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 120-127.
- Y. Desmedt, Threshold cryptography," European Transactions on Telecommunications, vol. 5, no. 4, pp. 449-457, 1994.
- Y. Deswarte, N. Abghour, V. Nicomette and D. Powell, "An internet authorization scheme using smart card-based security kernels", in Int'l Conf. on Research in Smart Cards (E-smart 2001), (Cannes, France), Lecture Notes in Computer Science, pp.71-82, Springer-Verlag, 2001.
- Y. Deswarte, L. Blain, J.-C. Fabre: Intrusion tolerance in distributed systems. In Proc. Symp. on Research in Security and Privacy, pages 110-121, Oakland, CA, USA, 1991. IEEE CompSoc Press.
- Durward McDonell, Brian Niebuhr, Brian Matt, David L. Sames, Gregg Tally, Szu-Chien Wang, Brent Whitmore. Developing a Heterogeneous Intrusion Tolerant CORBA System. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- Bruno Dutertre, Hassen Saïdi and Victoria Stavridou. Intrusion-Tolerant Group Management in Enclaves. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2001), Gotteborg, Sweden, 2001.
- J. Fraga and D. Powell, "A Fault and Intrusion-Tolerant File System", in IFIP 3rd Int. Conf. on Computer Security, (J. B. Grimson and H.-J. Kugler, Eds.), (Dublin, Ireland), Computer Security, pp.203-18, Elsevier Science Publishers B.V. (North-Holland), 1985.
- M. Kaâniche, E. Alata, V. Nicomette, Y. Deswarte, M. Dacier, Empirical analysis and statistical modeling of attack processes based on honeypots, WEEDS 2006 - Workshop on empirical evaluation of dependability and security, June 25 - 28, 2006, Philadelphia,USA

Design for Resilience

Further Reading

- R. Guerraoui, M. Hurn, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper, Consensus in asynchronous distributed systems: A concise guided tour," in Advances in Distributed Systems (S. Krakowiak and S. Shrivastava, eds.), vol. 1752 of LNCS, pp. 33-47, Springer, 2000.
- R. Guerraoui and M. Vukolic. Refined quorum systems. In Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems, pages 8–12, 2007.
- V. Gupta and V. Lam and H. Ramasamy and W. Sanders and S. Singh, Dependability and Performance Evaluation of Intrusion-Tolerant Server Architectures, Proceedings of the First Latin-American Symposium, 2003
- V. Hadzilacos and S. Toueg, Fault-tolerant broadcasts and related problems," in Distributed Systems (S. J. Mullender, ed.), New York: ACM Press & Addison-Wesley, 1993. An expanded version as Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca NY, 1994.
- HariGovind V Ramasamy, Prashant Pandey, James Lyons, Michel Cukier, William H. Sanders. Quantifying the Cost of Providing Intrusion Tolerance in Group Communication Systems, In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- Matti A. Hiltunen, Richard D. Schlichting and Carlos A. Ugarte. Enhancing Survivability of Security Services Using Redundancy. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.
- K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, The SecureRing protocolsfor securing group communication," in Proc. 31st Hawaii Int'l Conf. on System Sciences, pp. 317-326, IEEE, Jan. 1998.
- J. H. Lala, "A Byzantine Resilient Fault-Tolerant Computer for Nuclear Power Plant Applications", in 16th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-16), (Vienna, Austria), pp.338-43, IEEE Computer Society Press, 1986.
- B. Liskov and R. Rodrigues. Tolerating Byzantine faulty clients in a quorum system. In Proceedings of the 26th Int'l Conference on Distributed Computing Systems, June 2006.
- B. Madan, K. Goseva-Popstojanova, K. Vaidyanathan, K. Trivedi. Modeling and Quantification of Security Attributes of Software Systems. In Procs. of the Int'l Conf. on Dep. Syst. and Networks (DSN-2002), Washigton, USA, 2002.

© 2002-08 Paulo Veríssimo - All rights reserved, no unauthorized reproduction in any form

Further Reading

- D. Malkhi and M. K. Reiter, An architecture for survivable coordination in large distributed systems," IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 2, pp. 187-202, 2000.
- D. Malkhi and M. Reiter. Byzantine quorum systems. Distributed Computing, 11(4):203–213, 1998.
- Jean-Philippe Martin, Lorenzo Alvisi, Michael Dahlin. Small Byzantine Quorums. Procs. of Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- J. P. Martin and L. Alvisi. Fast Byzantine consensus. IEEE Transactions on Dependable and Secure Computing, 3(3):202–215, 2006.
- Roy A. Maxion and Tahlia N. Townsen, Masquerade Detection Using Truncated Command Lines. In Procs. of the Int'l Conf. on Dep. Syst. and Networks (DSN-2002), Washington, USA, 2002.
- F. Meyer and D. Pradhan, "Consensus with Dual Failure Modes," presented at The 17th Int'l Symp. on Fault-Tolerant Computing Systems, Pittsburgh, PA, 1987, pp. 214--22.
- L. E. Moser, P. M. Melliar-Smith, and N. Narasimhan. The SecureGroup communication system. Procs of IEEE Information Survivability Confer., pages 507–516, January 2000.
- Peter G. Neumann, "Practical Architectures for Survivable Systems and Networks," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report http://www.csl.sri.com/~neumann/private/arldraft.{pdf|ps}, October 1998.
- Nuno Ferreira Neves, João Antunes, Miguel Correia, Paulo Veríssimo, Rui Neves, *Using Attack Injection to Discover New Vulnerabilities*, Proceedings of the Int'l Conference on Dependable Systems and Networks (DSN), Philadelphia, USA, pp. 457-466, June 2006
- N.F. Neves, M. Correia, P. Veríssimo, Solving Vector Consensus with a Wormhole. IEEE Trans. Parallel and Distr. Systems, vol. 16, no. 12, pp. 1120-1131, Dec. 2005.
- S. Panjwani, S. Tan, K. Jarrin, and M. Cukier, *An Experimental Evaluation to Determine if Port Scans are Precursors to an Attack*, in Proc. *Int'l Conf. on Dependable Systems and Networks (DSN-2005)*, Yokohama, Japan, June 28-July 1, 2005, pp. 602-611
- P. Pal, F. Webber, and R. Schantz. The DPASA survivable JBI–a high-water mark in intrusion-tolerant systems. In Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems, pages 33–37, 2007.

Further Reading

- P. Porras, D. Schnackenberg, S. Staniford-Chen and M. Stillman, "The Common Intrusion Detection Framework Architecture", CIDF working group, http://www.gidos.org/drafts/architecture.txt, (accessed: 5 September, 2001).
- D. Powell, G. Bonn, D. Seaton, P. Veríssimo and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", in 18th IEEE Int. Symp. on Fault-Tolerant Computing Systems (FTCS-18), (Tokyo, Japan), pp.246-51, IEEE Computer Society Press, 1988.
- D. Ramsbrock, R. Berthier, M. Cukier, Profiling Attacker Behavior Following SSH Compromises, Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages: 119-124, 2007
- M. Reiter: Distributing trust with the Rampart toolkit; Comm's of the ACM,39/4, 1996.
- F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial", ACM Computing Surveys, 22 (4), pp.299-319, 1990.
- Paulo Sousa, Nuno Ferreira Neves, Paulo Veríssimo, How Resilient are Distributed f Fault/Intrusion-Tolerant Systems?. In Proc's of the Int'l Conference on Dependable Systems and Networks (DSN'05). Yokohama, Japan, pages 98-107, June 2005.
- P. Verissimo, A. Casimiro and C. Fetzer, "The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness", in Proc. of DSN 2000, the Int. Conf. on Dependable Systems and Networks, pp.533-52, IEEE/IFIP, 2000.
- Paulo Veríssimo, Nuno~Ferreira Neves, and Miguel Correia. The middleware architecture of MAFTIA: A blueprint. In Proceedings of the IEEE Third Information Survivability Workshop (ISW-2000), Boston, Massachusetts, USA, October 2000.
- Paulo Veríssimo, Travelling through Wormholes: a new look at Distributed Systems Models, SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), vol. 37, no. 1, (Whole Number 138), 2006.
- P. Veríssimo, Uncertainty and Predictability: Can they be reconciled?, Future Directions in Distributed Computing, pp. 108-113,Springer Verlag LNCS 2584, May, 2003
- Chenxi Wang, Jack Davidson, Jonathan Hill and John Knight. Protection of Software-Based Survivability Mechanisms. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.

Further Reading

- Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. 7th ACM Conference on Computer and Communications Security, Athens, November 2000, ACM Press, New York 2000, 245-254.
- L. Zhou, F. B. Schneider, and R. van Renesse, COCA: A secure distributed online certification authority,Tech. Rep. 2000-1828, CS Dpt, Cornell University, Dec. 2000. Also ACM TOCS to appear.
- S. Bhatkar, R. Sekar and D. C. DuVarney. Efficient techniques for comprehensive protection from memory error exploits. In Procedings of the 14th USENIX Security Symposium, pages 271-286, Aug. 2005.
- R. R. Obelheiro, A. N. Bessani, L. C. Lung and M. Correia. How practical are intrusion-tolerant distributed systems? DI/FCUL TR 06-15, Department of Informatics, University of Lisbon, Sep. 2006.
- P. Sousa, N. F. Neves and P. Verissimo. On the resilience of intrusion-tolerant distributed systems. DI/FCUL TR 06-14, Department of Informatics, University of Lisbon, Sep. 2006.
- P. Sousa, N. F. Neves and P. Verissimo. Resilient state machine replication. In Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing (PRDC), pages 305-309, Dec. 2005.
- P. Sousa, N. F. Neves and P. Verissimo. Hidden problems of asynchronous proactive recovery. In 3rd Workshop on Hot Topics in Sys. Dependability (HotDep'07), June 2007.
- D. Wang, B. Madan, K. Trivedi, Security analysis of SITAR intrusion tolerance system, Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems, pages 23 - 32, 2003
- J. Yin, J. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In Proceedings of the 19th ACM Symposium on Operating Systems Principles, pages 253-267, Oct. 2003.
- L. Zhou, F. B. Schneider and R. V. Renesse. APSS: proactive secret sharing in asynchronous systems. ACM Transactions on Information and System Security, 8(3):259-286, 2005.
- P. Zielinski. Paxos at war. Technical Report UCAM-CL-TR-593, University of Cambridge Computer Laboratory, Cambridge, UK, June 2004.
- J.~Xu, A.~Romanovsky, and B.~Randell. Concurrent exception handling and resolution in distributed object systems. IEEE Trans. on Parallel and Distributed Systems, 10(11):1019--1032, 2000.

1

Introduction
to
Intrusion Tolerance

Brief topics
on
security & dependability

The failure of computers

- *Why do computers fail and what can we do about it?*

  [J. Gray]
- Because:
  - All that works, fails
  - We tend to overestimate our HW e SW--- that's called faith☺
- So:
  - We had better prevent (failures) than remedy
- Dependability is ...
  - that property of a computer system such that reliance can justifiably be placed on the service it delivers
- Why?
  - Because (faith notwithstanding) it is the scientific way to quantify, predict, prevent, tolerate, the effect of disturbances that affect the operation of the system

Does not get better with distribution

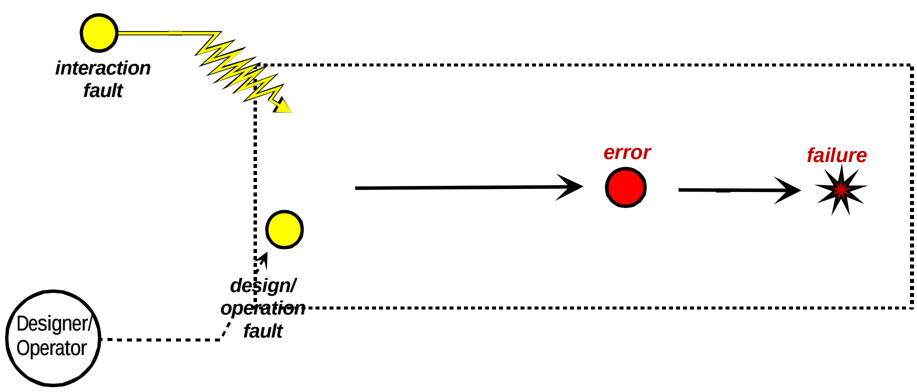
- *A distributed system is the one that prevents you from working because of the failure of a machine that you had never heard of.*

  [L. Lamport]
- Since:
  - Machines fail independently, for a start
  - But they may influence each other,
  - They communicate through unreliable networks, with unpredictable delays
- ...gathering machines renders the situation worse:
  - The reliability (<1) of a system is the product of the individual component reliabilities, for independent component failures
  - R(10 @ 0.99)= 0.9910= 0.90; R(10 @ 0.90)= 0.9010= 0.35

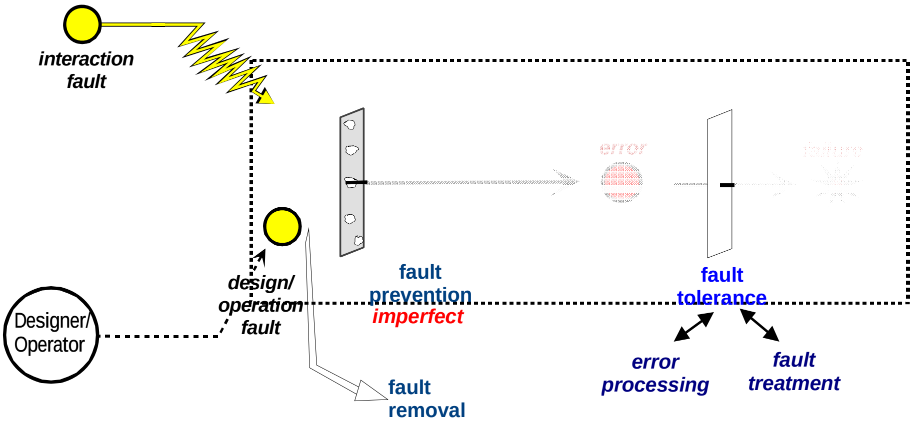
Can get much worse with malicious failures

- Failures are no longer independent
- Failures become more severe
- Fault models become less representative

sequence fault→ error→ failure

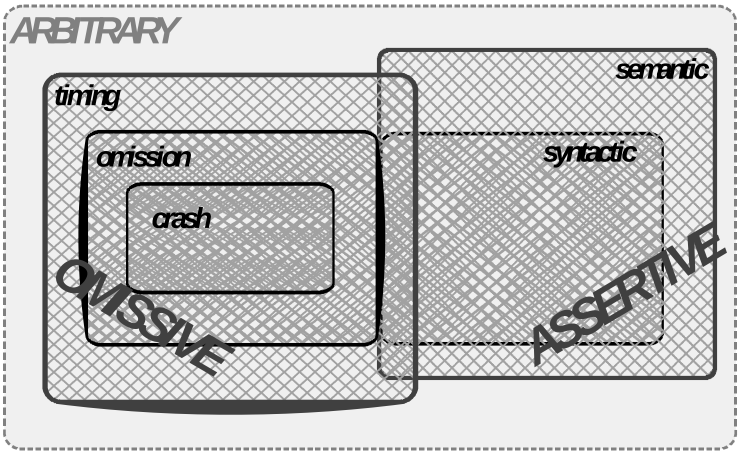


Dependability measures



Interaction Fault classification

- Omissive
  - Crash
    - host that goes down
  - Omission
    - message that gets lost
  - Timing
    - computation gets delayed
- Assertive
  - Syntactic
    - sensor says air temperature is 100°
  - Semantic
    - sensor says air temperature is 26° when it is 30°



Security Properties

- Confidentiality
  - the measure in which a service or piece of information is protected from unauthorized disclosure
- Integrity
  - the measure in which a service or piece of information is protected from illegitimate and/or undetected modification
- Authenticity
  - the measure in which a service or piece of information is genuine, thus protected from personification or forgery (*)
- Availability
  - the measure in which a service or piece of information is protected from denial of authorized provision or access

(*) also defined as a form of integrity of meta-information

Dependability properties

- Reliability
  - the measure of the continuous delivery of correct service (ex. MTTF)
- Maintainability
  - the measure of the time to restoration of correct service (ex. MTTR)
- Availability
  - measure of delivery of correct service with respect to alternation between correct and incorrect service (ex. MTBF/(MTBF+MTTR))
- Safety
  - the degree to which a system, upon failing, does so in a non-catastrophic manner

Some philosophy for a start

- What characterizes a dependable system?
  - A set of safety and liveness properties
- What characterizes a secure system?
  - A set of safety and liveness properties
- What may impair a dependable system?
  - A set of faults -> failure
- What may impair a secure system?
  - A set of faults (attacks, vulnerabilities, intrusions) -> failure
- How do I make a system dependable (normally)?
  - Using fault avoidance (prevention, removal) and fault tolerance (error detection, recovery, masking)
- How do I make a system secure (normally)?
  - Using fault avoidance (attack prevention, vulnerability removal)
  - and some bits of fault tolerance (intrusion detection)
  - Nowadays, increasingly fault tolerance (intrusion detection, recovery, masking)

Intrusion Tolerance

What is Intrusion Tolerance?

- The tolerance paradigm in security:
  - Assumes that systems remain to a certain extent vulnerable
  - Assumes that attacks on components or sub-systems can happen and some will be successful
  - Ensures that the overall system nevertheless remains secure and operational, with a measurable probability
- In other words:
  - Faults--- malicious and other--- occur
  - They generate errors, i.e. component-level security compromises
  - Error processing mechanisms make sure that security failure is prevented

Some preliminary observations...

Did you say trusted?

- Sometimes components are tamper-proof, others tamper-resistant...
  - Watch-maker syndrome:
    - --- *"Is this watch waterproof?"*
    - --- *"No, it's water-resistant"*
    - --- *"Anyway, I assume that I can swim with it!"*
    - --- *"Well...yes, you can... but i wouldn't trust that very much"*
- How can something trusted be not trustworthy?
  - *Unjustified reliance syndrome:*
    - --- *"I trust Alice"*
    - --- *"Well Bob, you shouldn't, she's not trustworthy"*
- What is the difference? If we separate *specification* from *implementation*, and provide a notion of *coverage*, all becomes clearer

Trust, Trustworthiness

- Trust
- the accepted dependence of a component, on a set of properties (functional and/or non-functional) of another component, subsystem or system
  - a trusted component has a set of properties that are relied upon by another component (or components).
  - if A trusts B, then A accepts that a violation in those properties of B might compromise the correct operation of A
- Trustworthiness
- the measure in which a component, subsystem or system meets a set of properties (functional and/or non-functional)
  - trustworthiness of B measures the *coverage* of the trust of A

Trusted vs. Trustworthy

- *Thou shalt not trust non-trustworthy components!*
- B is Trustworthy in the measure of the coverage with which its assumed properties are met... and coverage is never 1 in real systems...
- B should be Trusted only to the extent of its trustworthiness
  - trust may have several degrees, quantitatively or qualitatively
  - related not only with security-relat. properties (e.g., timeliness)
  - trust and trustworthiness lead to complementary aspects of the design and verification process
- we should talk about trusted-trustworthy components

Tamperproofness and its coverage or "tamper-resistance" not needed

- Tamperproof
  - Property of a system/component of being shielded, i.e. whose attack model is that attacks can only be made at the regular interface
  - Coverage of the "tamperproof" assumption may not be perfect, and there can be several degrees of such tamperproofness
- Example:
  - *Implementation of a security service using Java Cards to store private keys. We assume J.Cards are tamperproof, and so we argue that they are trustworthy (they will not reveal these keys to an unauthorised party). Hence we can justifiably argue that the service is trusted, with the coverage given by our assumptions, namely, the tamperproofness of JCards*

Intrusion Tolerance

terminology and concepts

Fault Models
Methodologies
Error processing
Fault treatment

Attacks, Vulnerabilities, Intrusions

- Intrusion
  - an externally induced, intentionally malicious, operational fault, causing an erroneous state in the system

an intrusion has two underlying causes:

- Vulnerability
  - malicious or non-malicious weakness in a computing or comm's system that can be exploited with malicious intention
- Attack
  - malicious intentional fault introduced in a computing or comm's system, with the intent of exploiting a vulnerability in that system

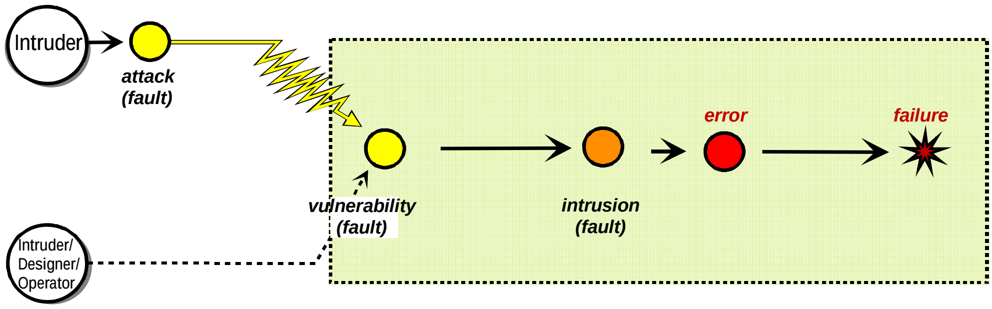
interesting corolaries:

- without attacks, vulnerabilities are harmless
- without vulnerabilities, there cannot be successful attacks

Attack-Vulnerability-Intrusion composite fault model

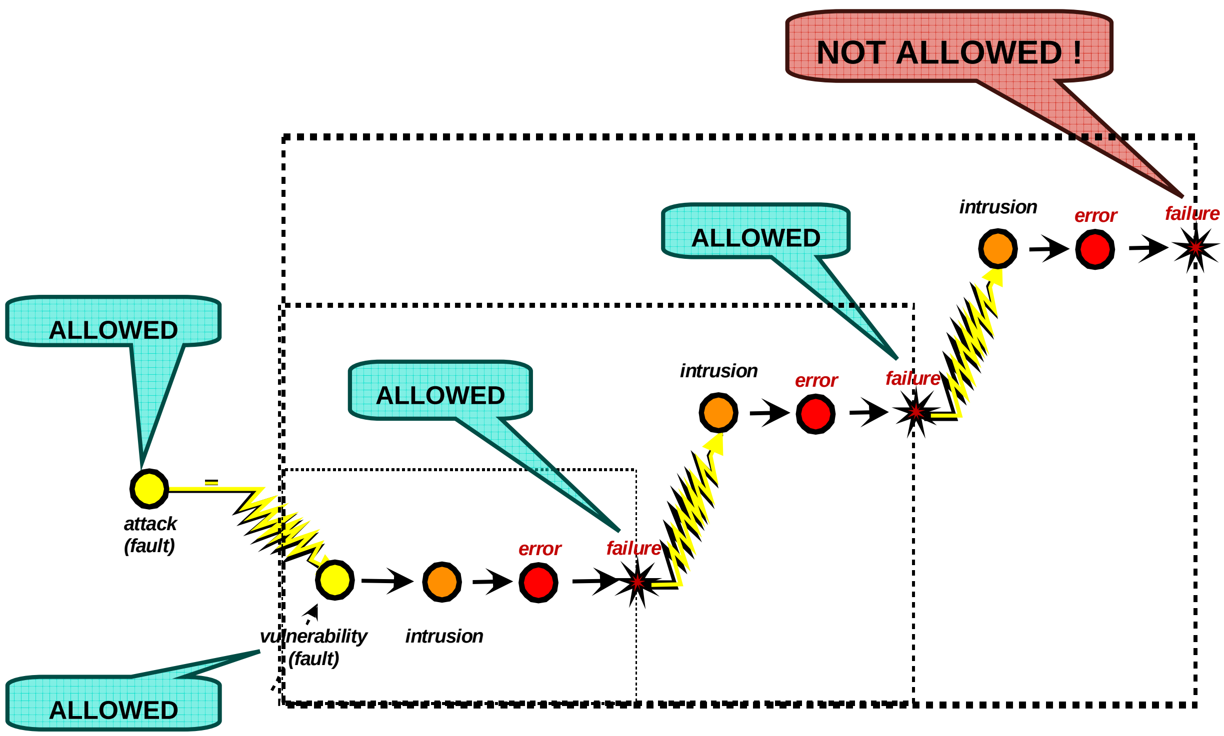
Hence: attack + vulnerability → intrusion → error → failure

A specialization of the generic *"fault,error,failure"* sequence



AVI sequence : *attack + vulnerability→ intrusion → error → failure*

Cascading Faults through error propagation

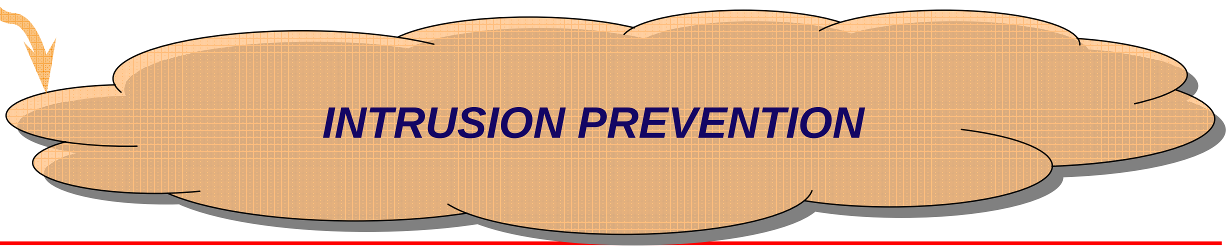


Intrusion Tolerance

Fault Models
Methodologies
Error processing
Fault treatment

Achieving trustworthiness w.r.t. malicious faults (the classical ways...)

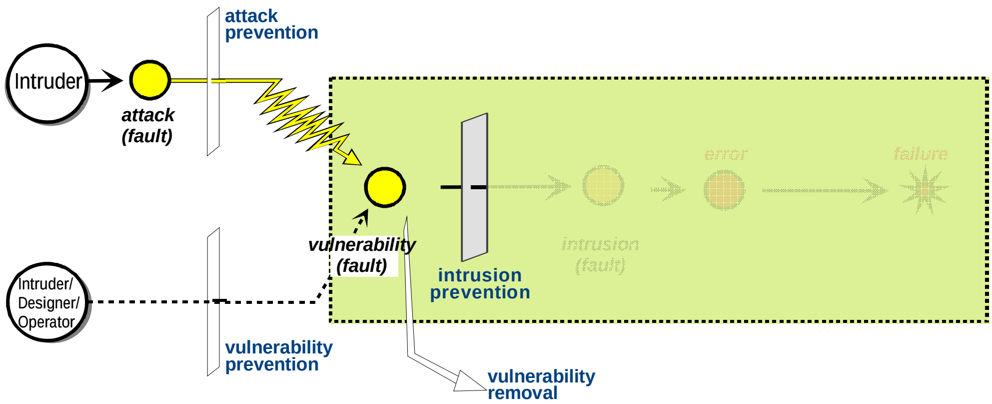
- Attack prevention
  - Ensuring attacks do not take place against certain components
- Attack removal
  - Taking measures to discontinue attacks that took place
- Vulnerability prevention
  - Ensuring vulnerabilities do not develop in certain components
- Vulnerability removal
  - Eliminating vulnerabilities in certain components (e.g. bugs)



Examples

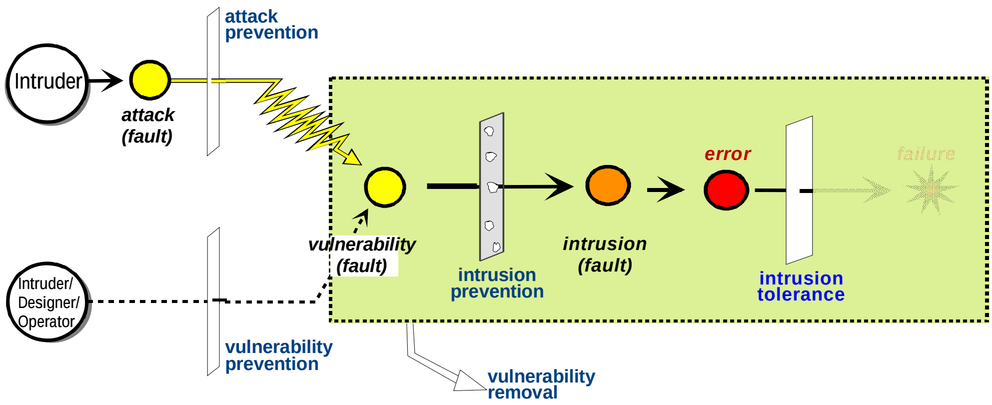
- Attack prevention
  - selectively filtering access to internal parts of the system (e.g., if a component is behind a firewall and cannot be accessed from the Internet, attack from there is prevented)
  - disabling JavaScript and/or Java prevents attacks by malicious scripts or applets
- Attack removal
  - identifying source of an external attack and taking measures to terminate it
- Vulnerability prevention
  - best practice in software development
  - measures preventing configuration and operation faults
- Vulnerability removal
  - of: coding faults allowing program stack overflow, files with root setuid in UNIX, naive passwords, unprotected TCP/IP ports

AVI Composite fault model



➢sequence : *attack + vulnerability→ intrusion→ failure*

AVI Composite fault model



- sequence : *attack + vulnerability→ intrusion→ failure*

Intrusion Tolerance

Fault Models

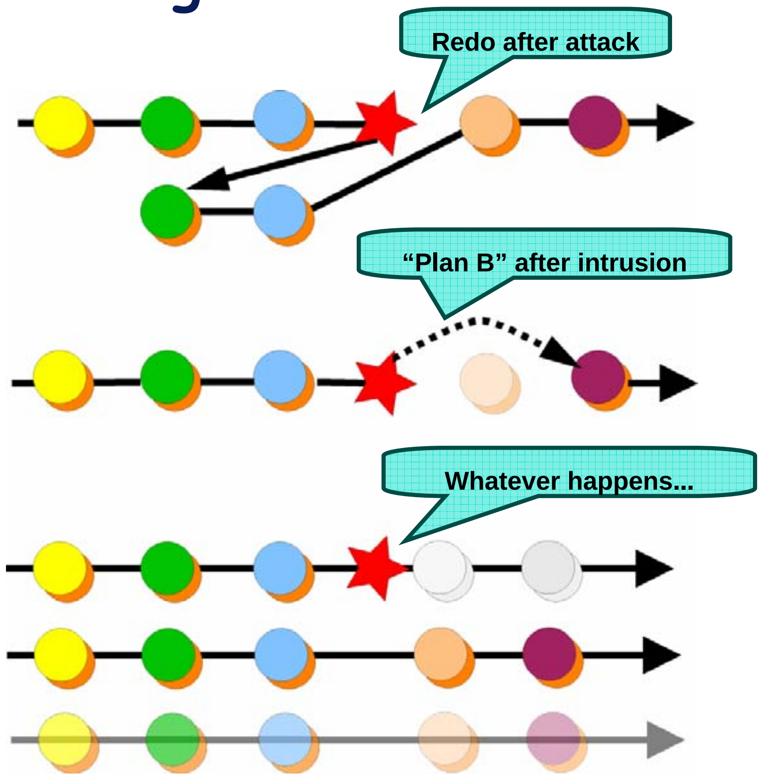
Methodologies

Error processing

Fault treatment

Error processing at work

- backward recovery
- forward recovery
- error masking



Intrusion Detection

Classical methodologies
ID as error detection
ID as fault diagnosis

ID: Error detection or fault diagnosis?

- classical IDS have two facets under intrusion tolerance
  - detecting errors as per the security policy specification
  - diagnosing faults as per the system fault model
- consider the following example:
  - *Organization A has an intranet with an extranet connected to the public Internet. It is fit with an IDS*
  - the IDS detects a port scan against an internal host, coming from the intranet
  - the IDS detects a port scan against one of the extranet hosts, coming from the Internet
  - *what is the difference?*

Intrusion Forecasting

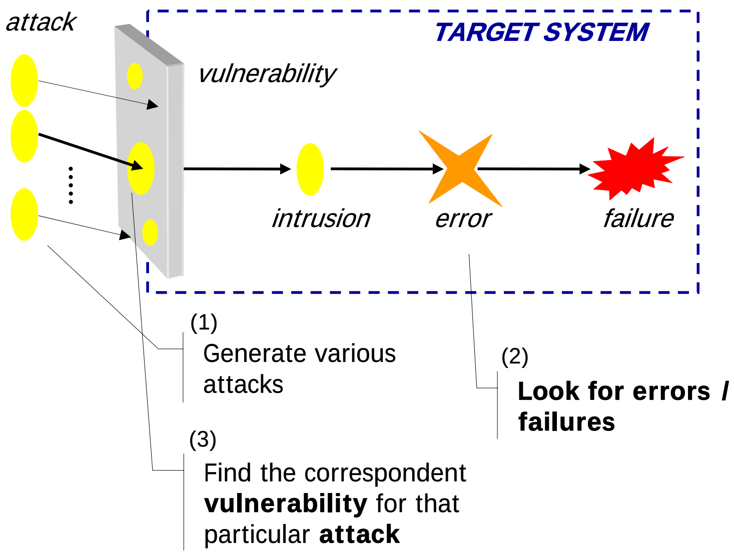
Attack injection
Vulnerability diagnosis
Assumption validation

Approaches

- Fault injection
- Static vulnerability analyzers
- Run-time prevention mechanisms
- Vulnerability scanners
- Fuzzers
- Attack injection -Using Attacks to Find Vulnerabilities

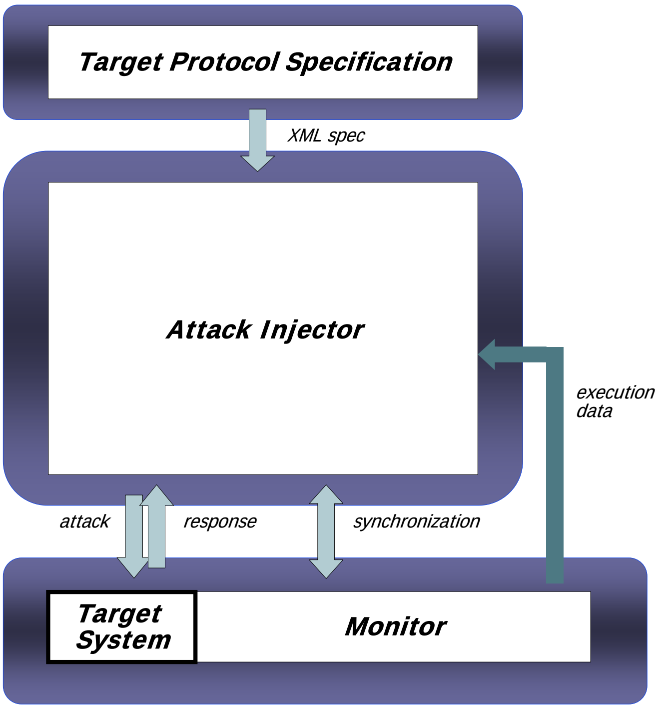
Using Attacks to Find Vulnerabilities

- Composite fault model AVI (Attack, Vulnerability, and Intrusion)



Attack Injection Tool

- Architecture



Design for Resilience

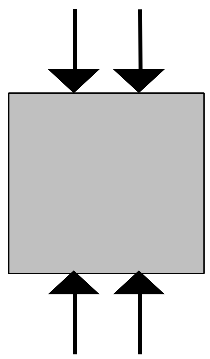
A biologically inspired metaphor of intrusion tolerance

Courtesy Christian Cachin, MAFTIA consortium

© 2002-08 Paulo Veríssimo - All rights reserved, no unauthorized reproduction in any form

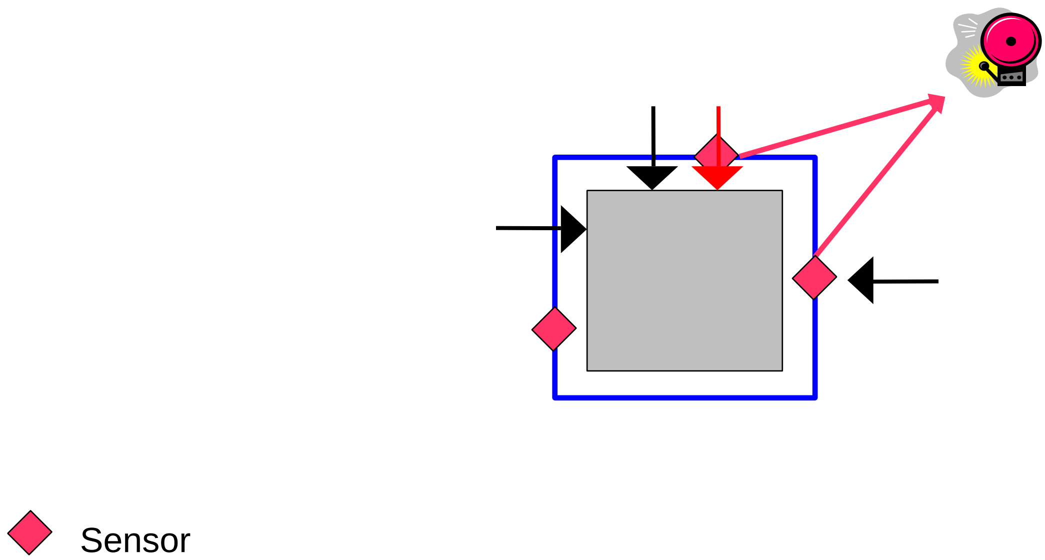
Computer system under attack

- no flaws, no vulnerabilities



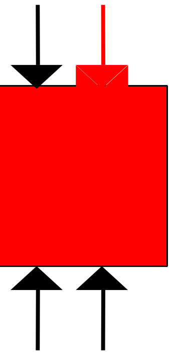
Intrusion detection

- Sensors for different attacks



Computer system under attack

- with vulnerabilities and
- successful attack

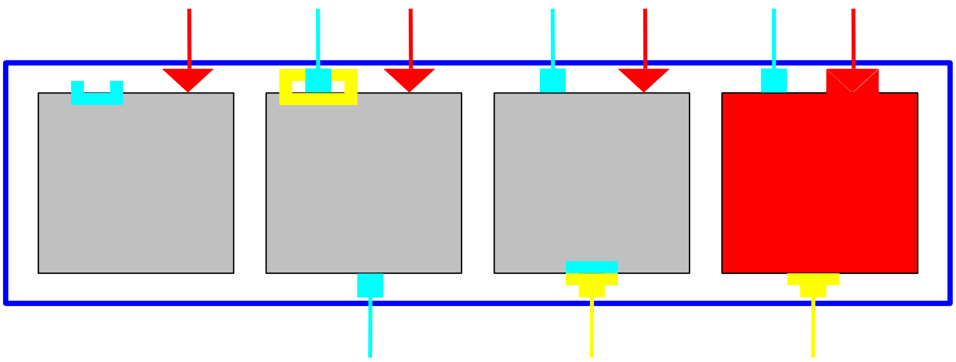


Vulnerability

Attack that exploits the vulnerability

Intrusion Tolerance

- with replicated and diverse structure
  - replicas have different vulnerabilities
  - majority remains intact

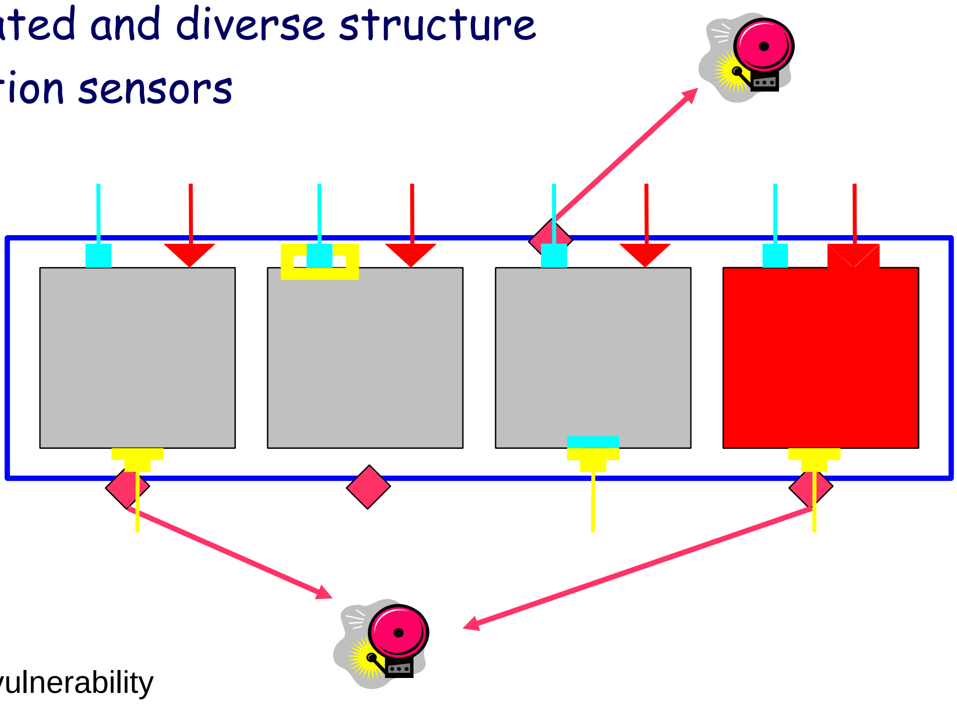


Vulnerability

Attack that exploits the vulnerability

Intrusion Tolerance and Detection combined

- with replicated and diverse structure
- with detection sensors



Vulnerability

Attack that exploits vulnerability

2

Resilience Building Paradigms

Resilience building paradigms

- Intrusion Detection
- Byzantine Failure Detection
- Self-enforcing vs.Trusted Third Parties
- Threshold cryptography
- Secret sharing
- Byzantine Reliable Broadcast
- Byzantine agreement
- Byzantine Consensus and Atomic Broadcast
- Byzantine State Machine Replication
- Quorums
- Fragmentation
- Randomisation
- Indulgence
- Separate execution and agreement
- Wormholes
- Reactive/Proactive recovery
- Diversity and obfuscation
- Proactive resilience

Resilience building paradigms

- Intrusion Detection

Intrusion Detection

Classical methodologies
ID as error detection
ID as fault diagnosis

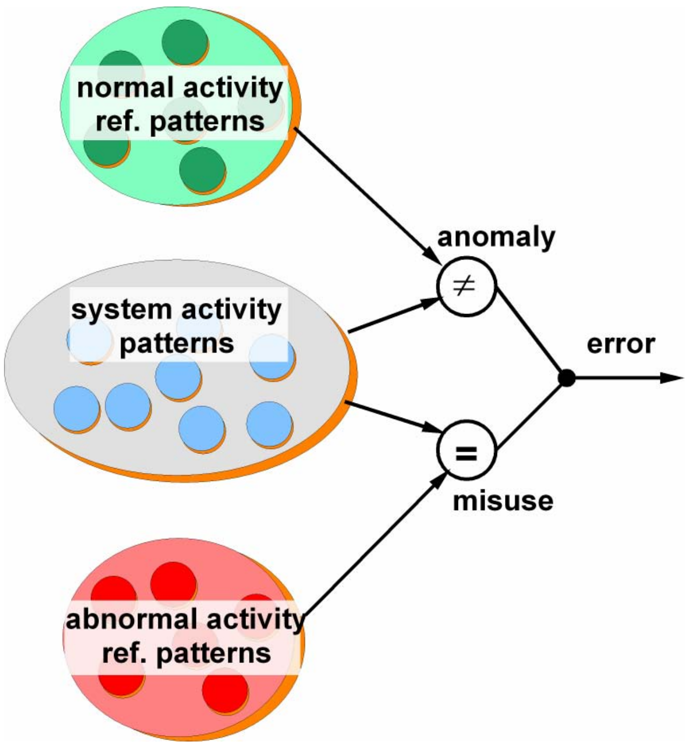
ID system classes

- Behavior-based (or anomaly detection) systems
  - no knowledge of specific attacks
  - provided with knowledge of normal behavior of monitored system, acquired e.g. through extensive training of the system
  - advantages: they do not require a database of attack signatures that needs to be kept up-to-date
  - drawbacks: potential false alarms; no info on type of intrusion, just that something unusual happened
- Knowledge-based (or misuse detection) systems
  - rely on a database of previously known attack signatures
  - whenever an activity matches a signature, an alarm is generated
  - advantage: alarms contain diagnostic information about cause
  - drawback: potential omitted or missed alarms, e.g. new attacks

Design for Resilience

Detection mechanisms

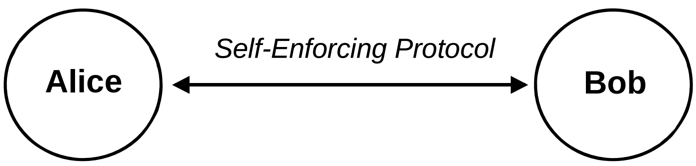
- consider system activity specified by patterns
- anomaly detection
  - looks for deviation from NORMAL ACTIVITY PATTERNS
- misuse detection
  - looks for existence of ABNORMAL ACTIVITY PATTERNS
- we can have hybrids
- Quality of Service
  - false alarm rate
  - omitted alarm rate



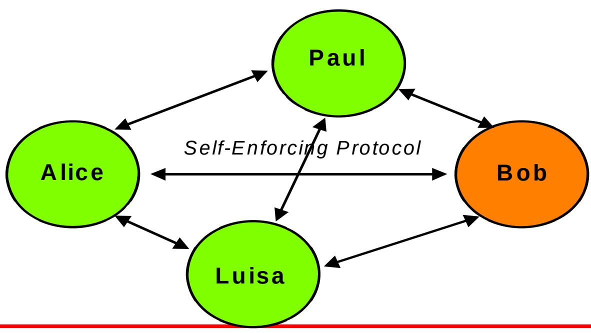
Resilience building paradigms

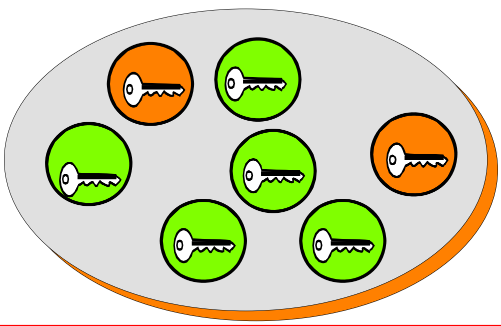
- **Self-enforcing vs. Trusted Third Parties**

Self-enforcing protocols



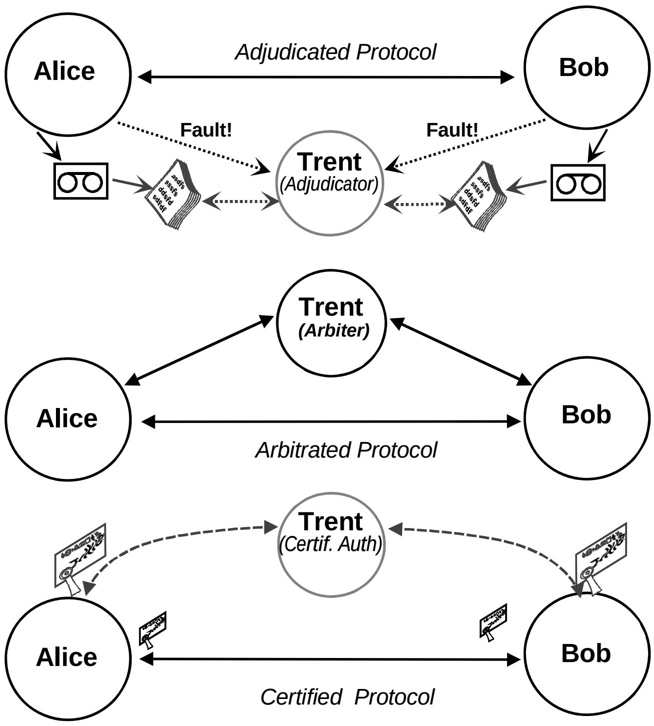
- Correct behaviour achieved by protocol participants alone
- They must build trust during protocol execution without trusting each other initially, and some maybe being malicious (e.g., by voting, k-out-of-n)





Trusted-Third-Party protocols

- Based on an apriori trusted component (TTP)
- TTP may be single point of failure
- adjudicated
  - Acting a posteriori if necessary to recover from errors
- arbitrated
  - Correct behaviour guaranteed during execution, errors prevented by arbiter
- certified
  - Correct behaviour guaranteed prior to execution through credentials supplied which limit participants misbehaviour during execution (errors prevented)



Resilience building paradigms

- **Threshold cryptography**
- **and secret sharing**

Threshold cryptography and secret sharing

- "Intrusion-tolerant" cryptography
- Given N processes each holding part of crypto secret

- **Secret sharing**:
  - Example a shared secret key
  - Any k-out-of-N processes combine their shares and reconstruct secret s
  - Any k-1 colluding or intruded processes cannot reconstruct s
- **Function sharing**:
  - Example a threshold signature
  - k processes together execute function F
  - k-1 colluding or intruded processes cannot execute F

Proactive secret sharing

- A process cannot know whether its share is "good"
- If one share is corrupted the secret is not reconstructed

- **Proactive secret sharing**
  - A period Tf is assumed as an estimate of time for f+1=k failures to be produced, e.g., to corrupt k processes
  - (these k processes would be able to get the secret)
  - Every Tss < Tf, protocol recalculates the shares (reconstructs) without changing the secret

Resilience building paradigms

- **Byzantine Reliable Broadcast/Multicast**
- **Byzantine agreement**

Basic failure modes

- Processes can fail in a Byzantine way:
  - Crash, disobey the protocol, send contradictory messages, collude with other malicious processes,...
- Network:
  - Can corrupt packets (due to accidental faults)
  - An attacker can modify, delete, and introduce messages in the network

Reliable multicast

- A reliable multicast protocol is defined formally in terms of the following properties:
- **Validity:** If a correct process multicasts a message M then some correct process in *group(M)* eventually delivers M.
- **Agreement:** If a correct process delivers a message M then all correct processes in *group(M)* eventually deliver M.
- **Integrity:** For any message M, every correct process *p* delivers M at most once and only if *p* is in *group(M)*, and if *sender(M)* is correct then M was previously multicast by *sender*(*M*).

Resilience building paradigms

- **Byzantine Consensus and Atomic Broadcast**

Consensus properties

- Validity
  - If all correct processes propose the same value v, then any correct process that decides, decides v
- Agreement
  - No two correct processes decide differently
- Termination
  - Every correct process eventually decide

- With Byzantine failures, *Validity* makes little sense
- Vector consensus improves the situation

- Consensus is equivalent to atomic broadcast

Vector Consensus properties

- Validity
  - Every correct process decides on a vector *vect* of size *n* such that:
  - 1. For every *1 =< i =< n*, if process *pi* is correct, then *vect[i]* is either the initial value of *pi* or the value bottom
  - 2. at least *f+1* elements of the vector *vect* are the initial values of correct processes.
- Agreement
  - No two correct processes decide differently
- Termination
  - Every correct process eventually decide

Atomic Broadcast properties

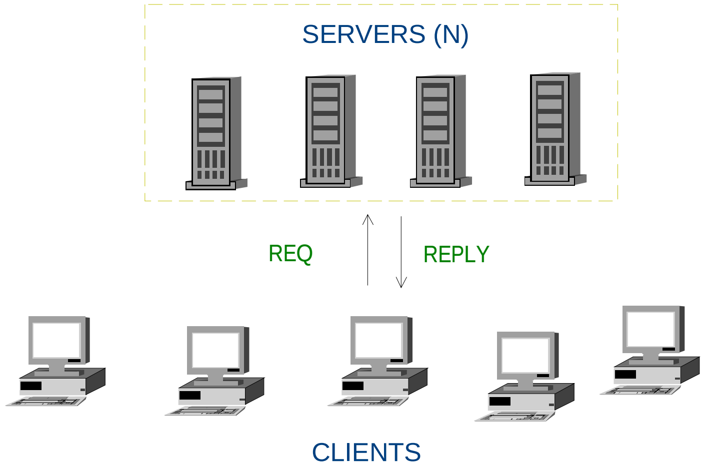
- Validity
  - If a correct processor multicasts a message M, then some correct processor eventually delivers M.
- Agreement
  - If a correct processor delivers a message M, then all correct processors eventually deliver M.
- Integrity
  - For any message M, every correct processor *p* delivers M at most once, and if *sender(M)* is correct then M was previously broadcast by *sender(M)*.
- Total order
  - If two correct processors deliver two messages *M1* and *M2* then both processors deliver the two messages in the same order.

Resilience building paradigms

- **Byzantine State Machine Replication**

Byzantine State Machine Replication

- Rules:
  - they execute atomic commands, change state and produce outputs
  - commands are deterministic
- If:
  - servers start in same state
  - execute same sequence of inputs in same order
- Then,
  - all follow same sequence of state/outputs



Byzantine State Machine Replication

- input requirements:
  - commands delivered by Byzantine atomic broadcast protocol
- Failures of servers can be arbitrary
- given N number of servers, maximum number of servers that can fail is: $f = \lfloor \frac{N-1}{3} \rfloor$
- or in other words: $N \geq 3f + 1$

- this limit is actually imposed by the protocol used to disseminate messages (ABCAST)
  - ex: N=4 servers tolerate f=1 corrupt;
    N=7 tolerates f=2

Where do we go from here?

- arbitrary failures / asynchrony thread
  - are safe, but normally inefficient
  - FLP: no deterministic solution of hard problems e.g. consensus, BA, SMR with ABCAST
  - does not solve timed problems (e.g., e-com, stocks)

Arbitrary failure / asynchrony assumptions

- OBJECTIVE:
  - solve most non-timed problems with high coverage
- tone down determinism:
  - randomization (Maftia/IBMZurich/Cachin-et-al)
  - semantics (+) - speed (-)
- tone down liveness expectations:
  - sacrifice liveness guarantees (MIT/Castro-Liskov)
  - termination (-) - speed (+)
- use weaker semantics
  - avoid consensus (Cornell/APSS/Schneider-et-al)
  - use quorums (Alvisi, Malki, Reiter)
  - semantics (-) - termination (+)
- Coverage:
  - very high, but still bound to crucial assumptions, such as number of failures
- Timeliness:
  - none

Controlled failure assumptions

- OBJECTIVE:
  - solve non-timed problems with high coverage

- tone down fault severity:
  - hybrid faults (IBMZurich/Cachin-et-al) (Meyer, Pradhan, Walter, Suri)
  - fault coverage (~)
- enforce hybrid behaviour ("strong" and "weak" components):
  - architectural hybridization (U.Lisboa)
  - speed (+) - termination (+) - semantics (+)
  - fault coverage (+)

- Coverage:
  - fair for hybrid fault coverage
  - can get very high if bound to the "strong" components
  - still bound to crucial assumptions, such as nr of failures
- Timeliness:
  - none

Resilience building paradigms

- **Randomisation**

Randomisation

- Another way to overcome the asynchronous impossibility of determinism is to use a probabilistic approach to solve consensus
- It does not require any explicit or implicit timing assumptions
- These algorithms usually have a large number of excepted communication steps and/or rely heavily on public-key cryptography

High-performance Randomisation

- These features have led to a couple of general (wrong) beliefs about randomisation inefficiency:
  - too slow to be used in practice
  - local coin tossing slower than shared coin tossing
- But...two important points have been overlooked:
  - Consensus protocols are not executed in oblivion
  - The theoretical adversary models is not very realistic
- With this in mind, high-performance solutions were recently found, bringing new practicality to randomised consensus

Resilience building paradigms

- **Indulgence**

Indulgence

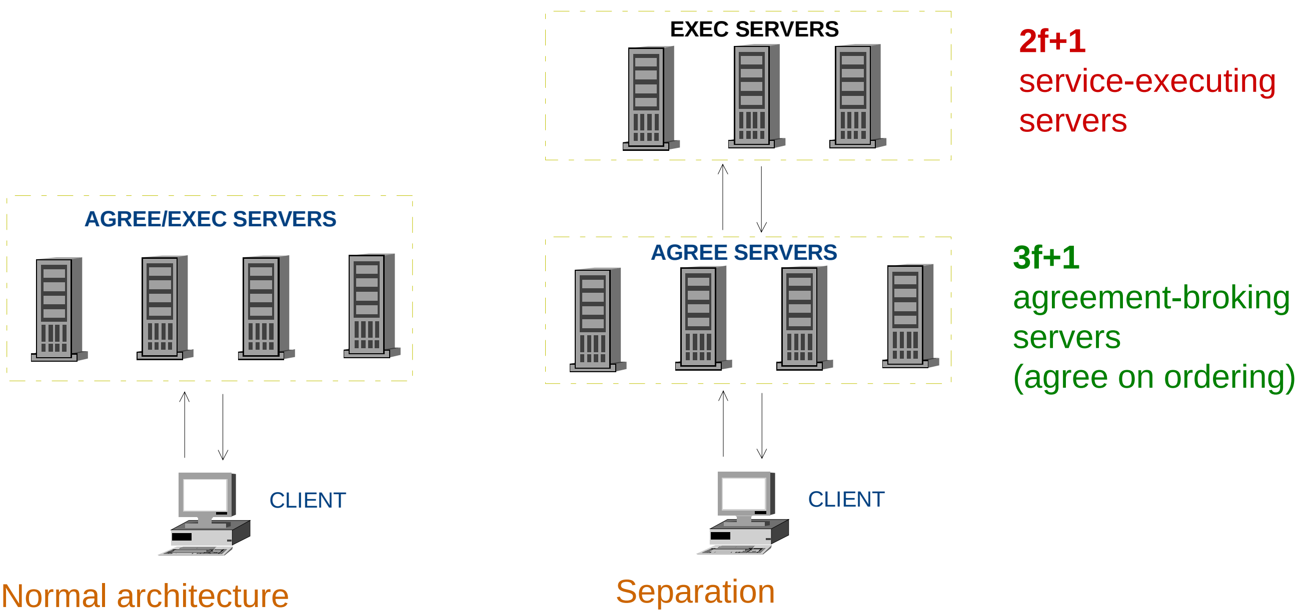
- Another way to overcome hardness of asynchronous non-determinism (FLP) is to:
  - allow protocols not to have liveness (i.e., not to terminate)
  - but guaranteeing that they always have safety
- This way, partial synchrony assumptions can be made in a safe way
  - if attacked, all that happens is that protocol stalls but never makes mistakes

Resilience building paradigms

- **Separate execution and agreement**

Separate execution and agreement [Yin et al.]

- Separate server sets allow to lower the number of required execution servers to **2f+1**, maintaining **3f+1** for agreement



Resilience building paradigms

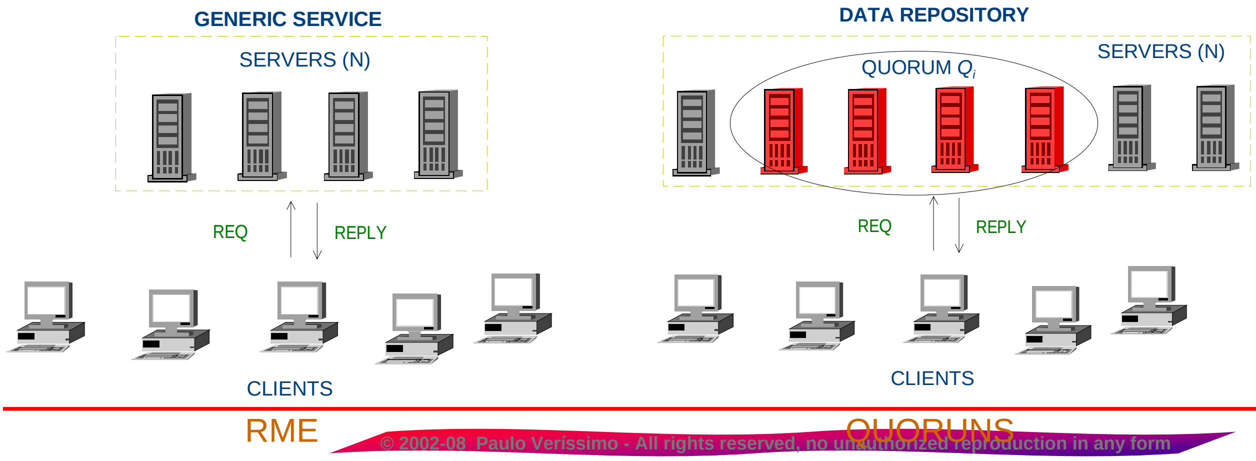
- **Quorums**

Quorums

- Another way to overcome limitations imposed by FLP in arbitrary failure modes
- a quorum system *Q* is a server set such that
- forall Q1, Q2 in *Q*, Q1 and Q2 always intersect
- operations are performed over a quorum

Quorums vs SM

- **SM** – generic solution for data and/or operations/services
- **Quorums** – dedicated to data repositories



Resilience building paradigms

- **Fragmentation**

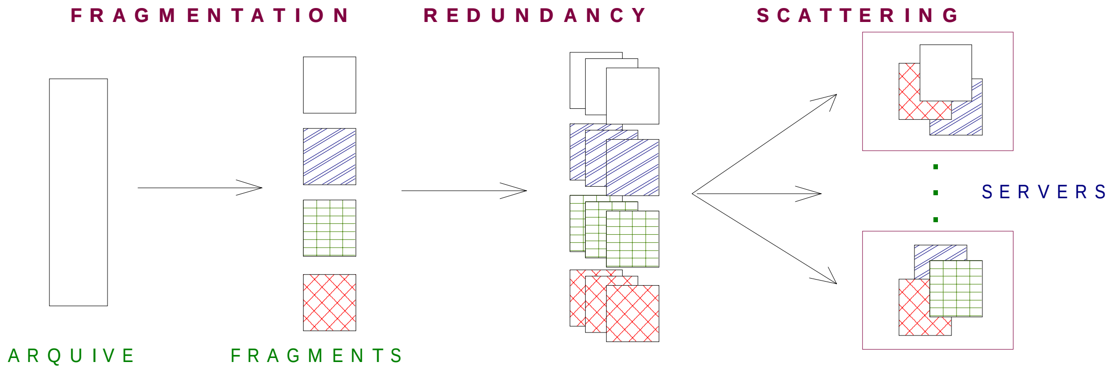
Quorums vs Fragmentation

- **Quorums:**
  - emphasis on small memory objects (variables, tuples)
- **Fragmentation:**
  - emphasis on large memory objects (files, archives)

Fragmentation, Redundancy, Scattering – FRS

[Fraga & Powell 85]

- Data endures three steps:
  - *fragmentation* – data is fragmented, confidenctiality is not perfect, but fragments yield practically nothing of whole
  - *redundancy* – fragments are replicated to tolerate losses
  - *scattering* – fragments are disseminated throughout system repositories



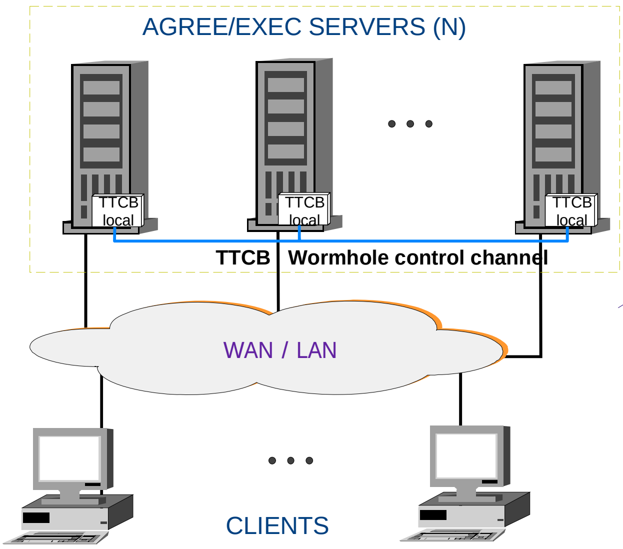
Resilience building paradigms

- **Wormholes**

Wormhole-aware

[M. Correia, N. Neves, P. Veríssimo], U. Lisboa

- Wormhole-aware server sets with a hybrid failure model allow to lower the number of required servers to **2f+1**



2f+1
agree/exec servers

WORMHOLE:
limited functionality
but trustworthy subsystem

Service is executed in servers with occasional calls to TTCB (agree on ordering)

Resilience building paradigms

- Exhaustion safety

Taking long detours...

- OBJECTIVE:
  - keep systems working long enough (non-timed problems, arbitrary failures / asynchrony thread)

- ensuring enough replicas
- using diversity and obfuscation

- OBJECTIVE:
  - keep systems working long enough or in a perpetual manner

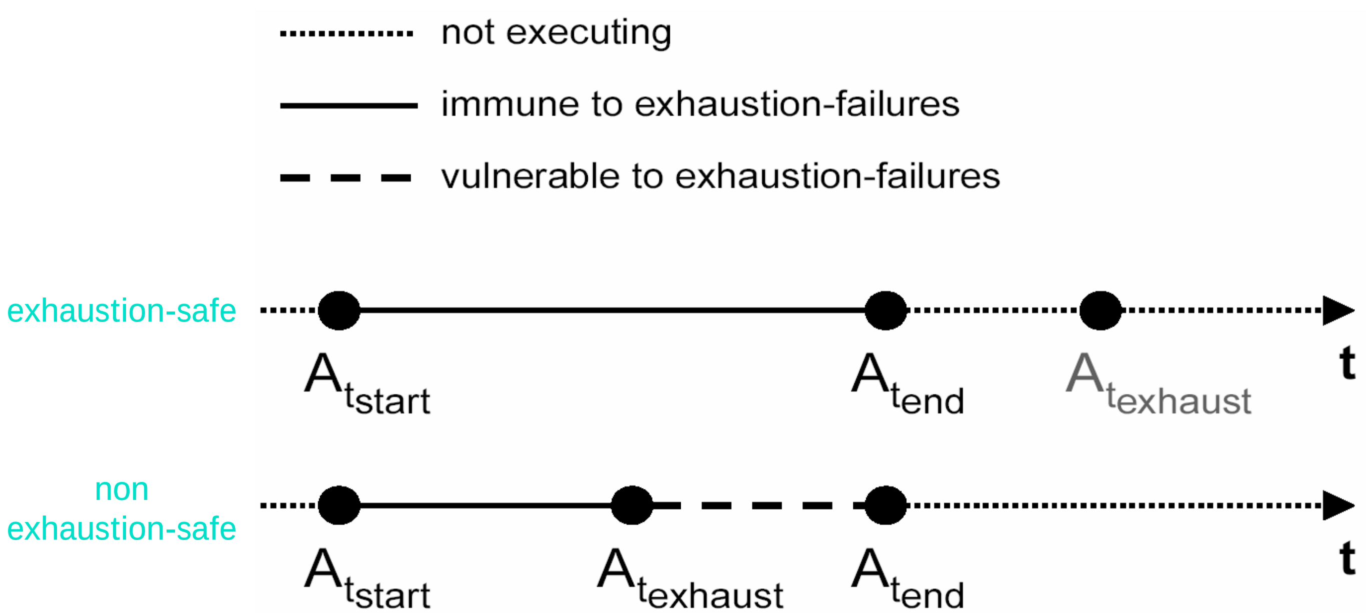
- reactive or proactive recovery (e.g., rejuvenation, refreshing)

Detours may lead to dead ends…

- f fault-tolerance means at least (n-f) correct nodes.
- Resource exhaustion: violation of a resource assumption (e.g., f+1 nodes fail), which may lead to failure
- An exhaustion-failure is a failure that results from resource exhaustion.

- A system is exhaustion-safe if resource exhaustion never happens.

To Be or Not to Be Exhaustion-Safe



Resilience building paradigms

- Reactive/Proactive recovery
- Diversity and obfuscation
- Proactive resilience

Async Proactive Recovery

- How to guarantee that rejuvenations always terminate before resource exhaustion?
  - Rejuvenation start instant may be delayed.
  - Rejuvenation actions may be delayed.
  - These delays may be enforced by a malicious adversary!
- Async proactive recovery does not guarantee exhaustion-safety.
  - namely, in a malicious environment.

Problems of proactive recovery

- Problems that may affect proactively recovered intrusion-tolerant systems:
  - 1. adversary may be more powerful than assumed
  - 2. adversary may slow down the pace of recovery
  - 3. adversary may perform stealth attacks on the system timing
  - 4. recoveries may reduce system availability

- Classical proactive recovery systems are affected by all 4
- *Proactive resilience* deals with problems 2, 3 and 4. (Problem 1 is fundamentally unsolvable) [Sousa et al., SAC06]

Detours may lead to dead ends…

- An f fault-tolerant distributed system is exhaustion-safe if it terminates before f+1 faults being produced
- Obvious?
- *Impossibility of exhaustion-safe asynchronous distributed systems (w/ or w/o proactive recovery)*

Proactive resilience

[Sousa, Verissimo, Neves]

- Combining proactive recovery and wormholes
  - Proactive recovery is useful to postpone $t_{exhaust}$ as long as it has timeliness guarantees.
  - Proposal: combine async payload system with sync proactive recovery subsystem.

Limitations of some current IntTol paradigms

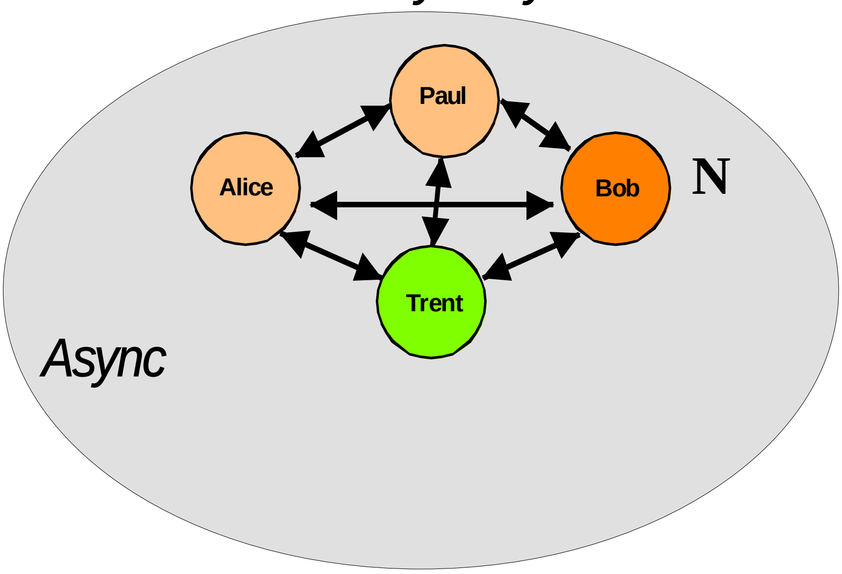
Limitations of IntTol paradigms

- Resource exhaustion unnoticed

or

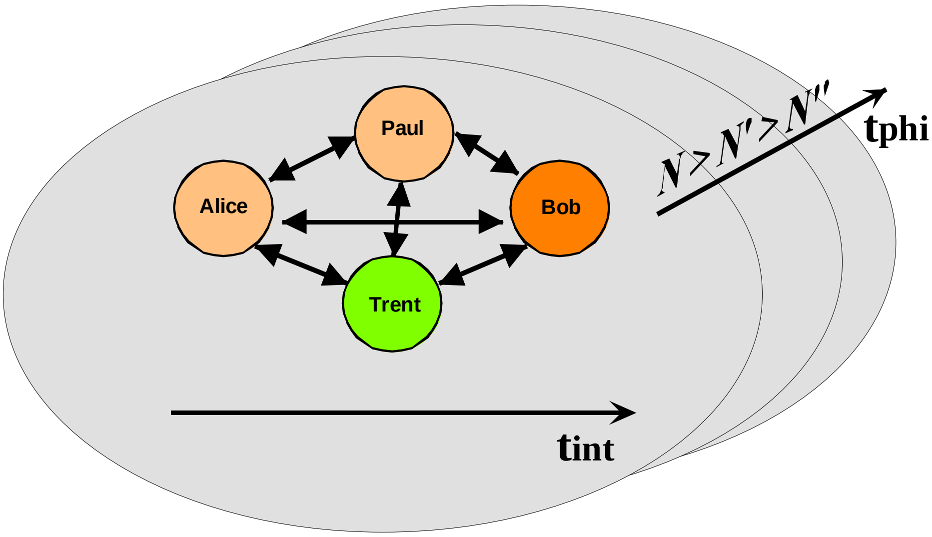
- why attackers work in real time

Classical Model - Async System



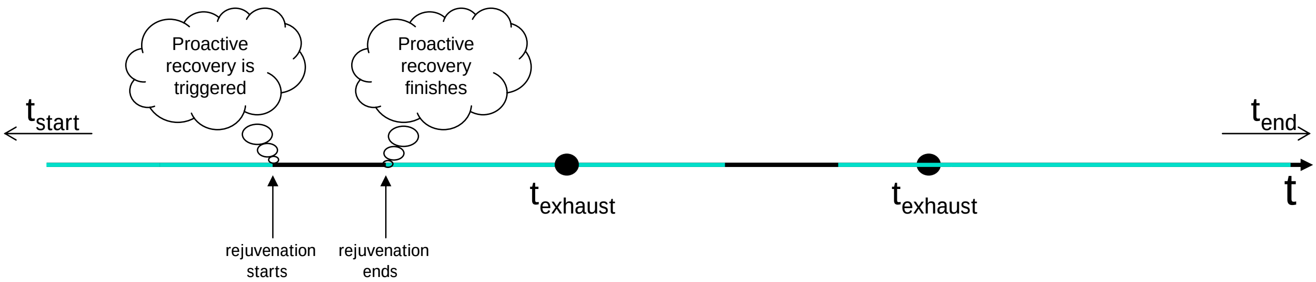
Burn like a candle or.... Burn like a match?

Classical Model vs. Reality



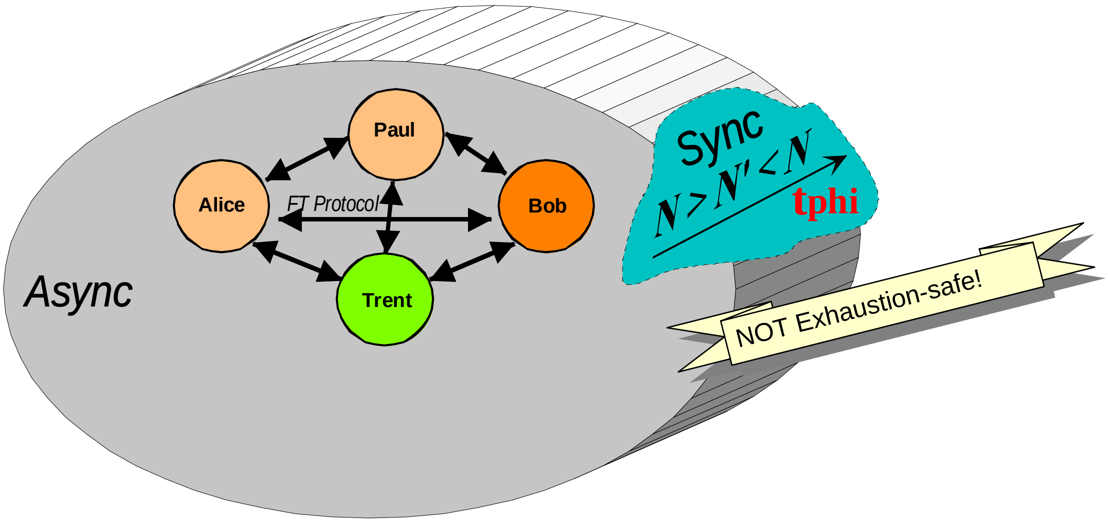
Proactive Recovery

- Goal: to constantly postpone $t_{exhaust}$ through periodic rejuvenation.
  - e.g., periodic rejuvenation of secret keys, OS code, etc .



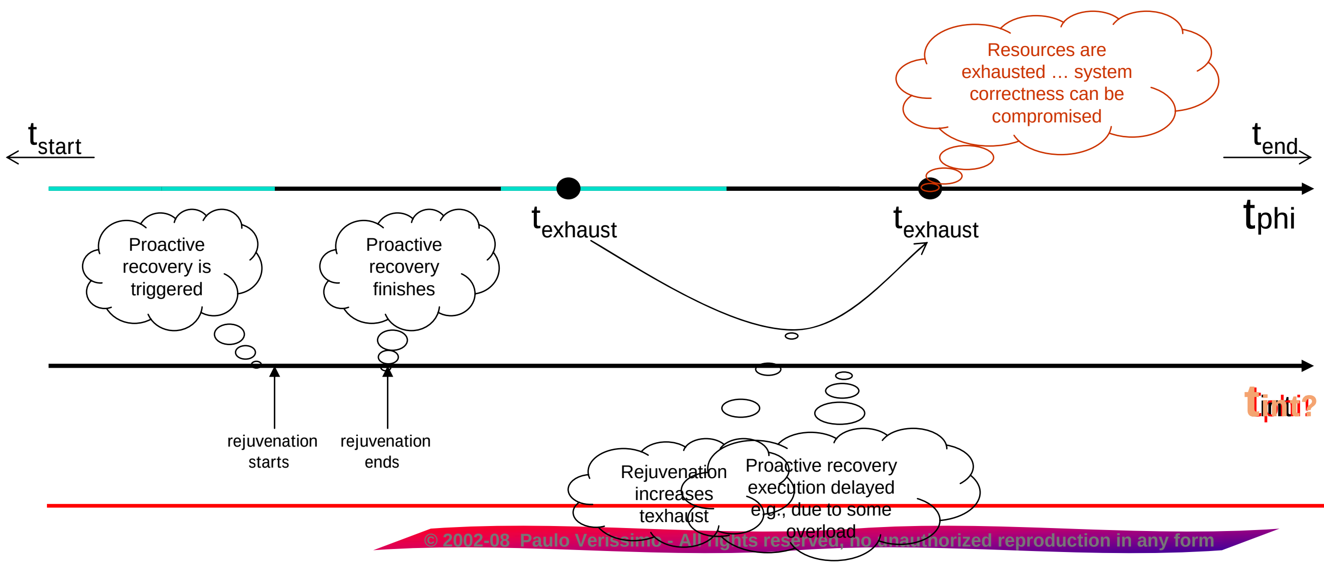
- A system is exhaustion-safe only if rejuvenations are always terminated before exhaustion.

Physical Model - shows Async system with unaccounted for synchrony assumptions



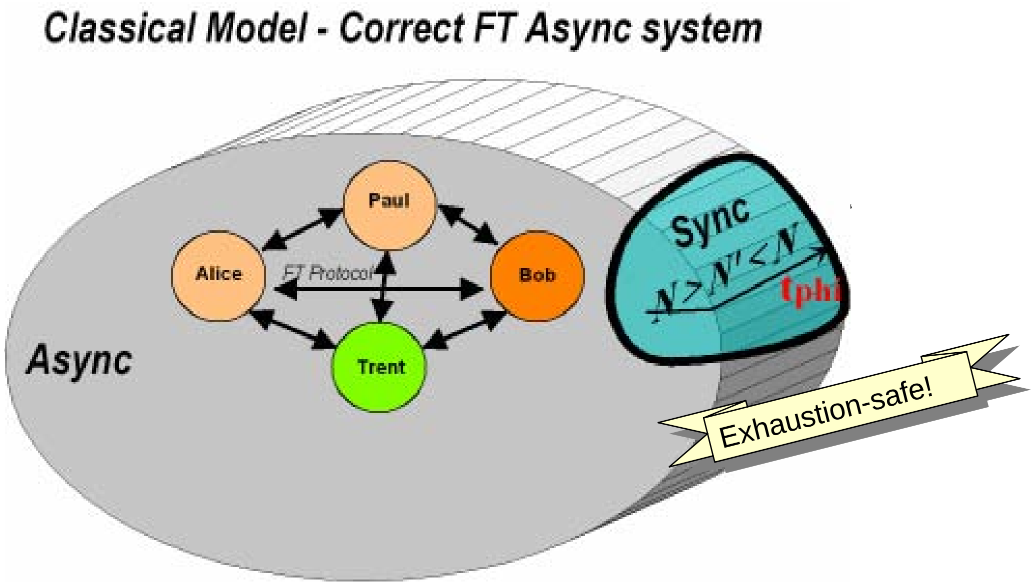
Proactive Recovery

- Goal: to constantly postpone $t_{exhaust}$ through periodic rejuvenation.
  - e.g., periodic rejuvenation of OS code .



The case for hybrid dis sys models

- in an asynchronous system, all synchrony must be encapsulated



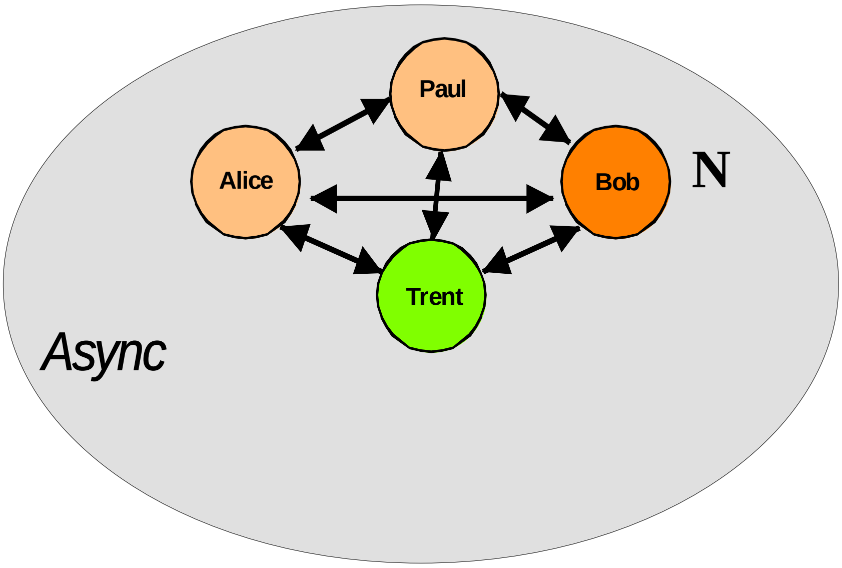
Limitations of IntTol paradigms

- **Homogenous models and hidden assumptions**

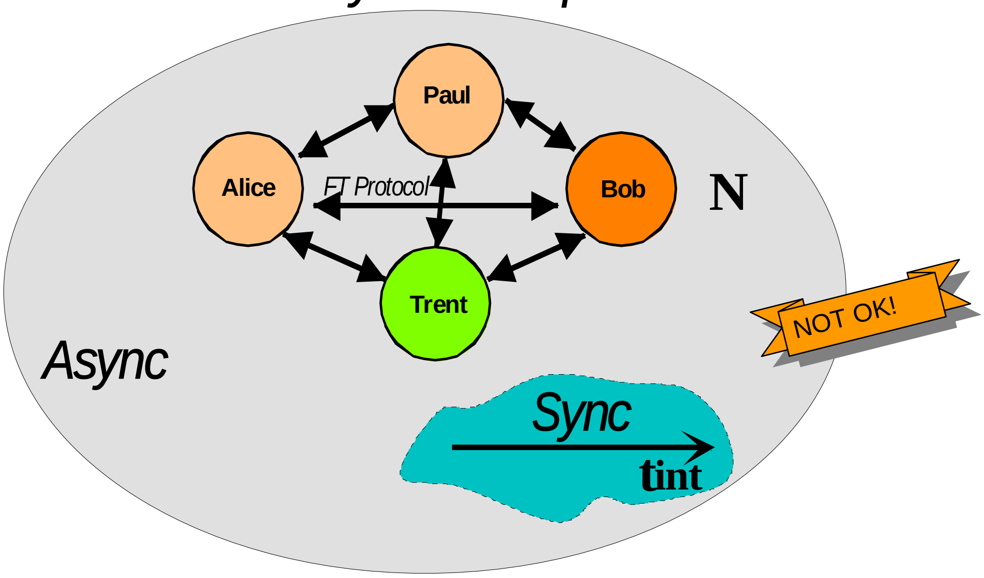
or

- **why attackers pick the weakest link**

Classical Model - Async System



Classical Model - Async System with hidden sync assumptions



From Theory to Practice (1)

- System Model:
  - async model, malicious adversary.
  - private key shared by servers using threshold cryptography.
  - Shares are periodically refreshed through an asynchronous proactive secret sharing protocol (APSS).
  - Key is compromised if an adversary collects sufficient shares in the interval between successive executions of the APSS.
- Algorithmic assumptions:
  - n servers share the private key using (n, f+1) secret sharing scheme
  - f+1 shares are sufficient to recover the key.
  - less than f+1 shares give no knowledge about the key.
  - At most $f \leq (n-1)/3$ servers "are compromised at any time".
    - Excludes the possibility of an adversary controlling f+1 servers simultaneously,
    - but "does not rule out learning f+1 shares one at a time" (mobile virus attack)

The problem

- when safety of an asynchronous system depends on non-substantiated timing assumptions
    - clocks with bounded rate of deviation to real-time
    - capacity of performing periodic (timely) executions
    - these assumptions can be violated either in the assumed async environment and/or by a malicious adversary.

From Theory to Practice (4)

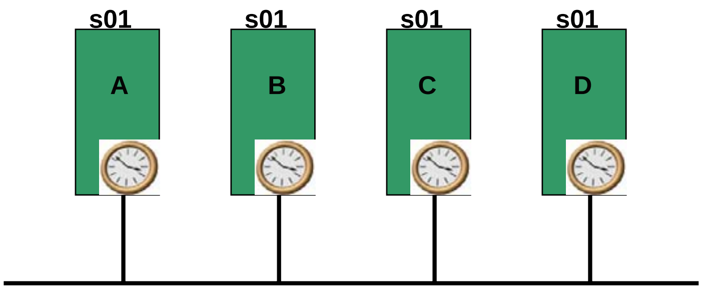
- An attack that compromises safety:
  - Two adversaries: ADV1 and ADV2.
  - Step 1: ADV1 performs a mobile virus attack against f+1 servers
    - slows the clock rate of each server.
  - Step 2: ADV1 temporally cuts off the links between the f+1 servers and the rest of the system.

From Theory to Practice (5)

- An attack that compromises safety:
  - Step 3: ADV2 performs a mobile virus attack against the same f+1 servers
    - learns, one by one, f+1 private key shares.
    - no rejuvenation occurs in between because in step 1 clocks are made as slow as needed.
  - Step 4: ADV2 discloses private key by combining the f+1 shares.
  - Important Note: ADV1 actions simply enforce a behavior that can occur in any fault-free async system.

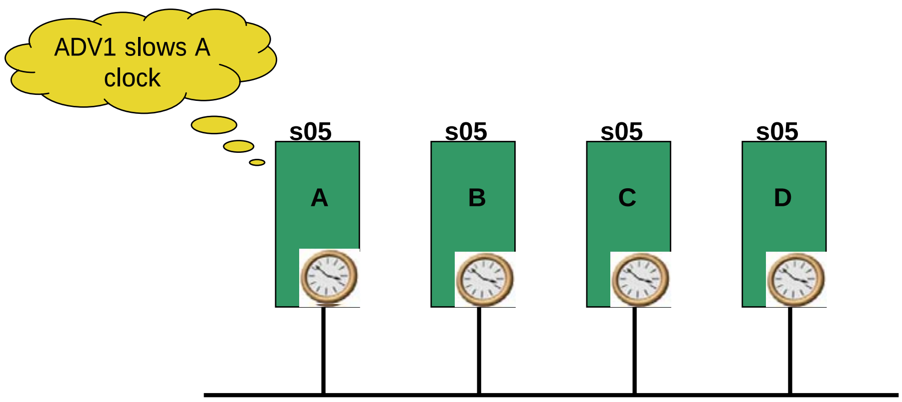
From Theory to Practice (6)

- Example with n=4, f=1



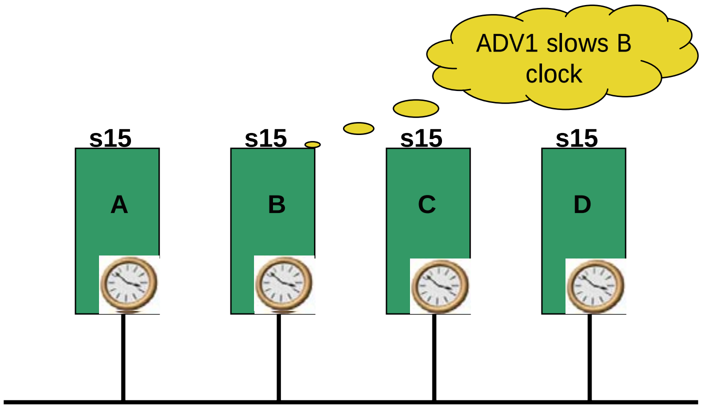
From Theory to Practice (6)

- Example with n=4, f=1



From Theory to Practice (6)

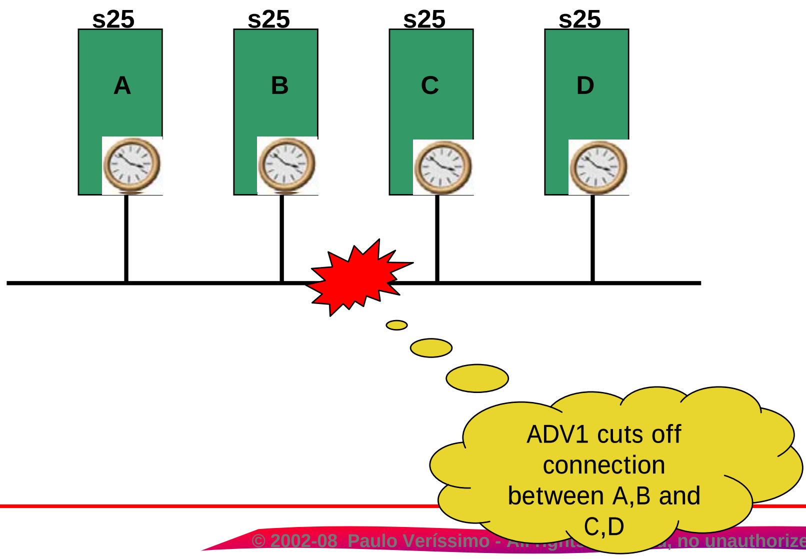
- Example with n=4, f=1



Design for Resilience

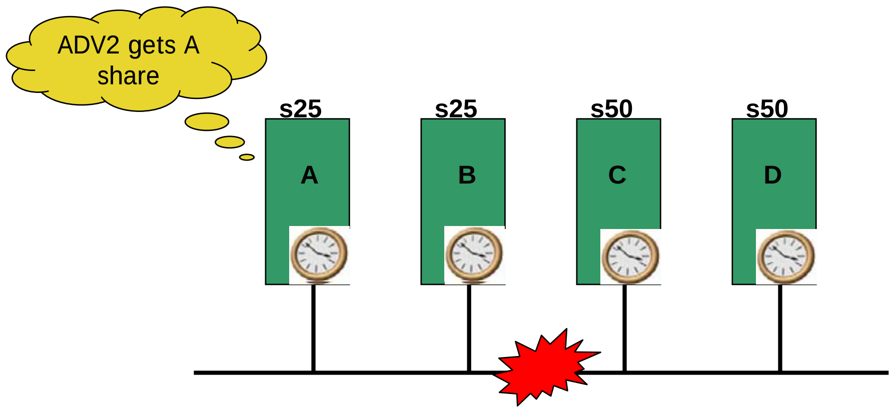
From Theory to Practice (6)

- Example with n=4, f=1



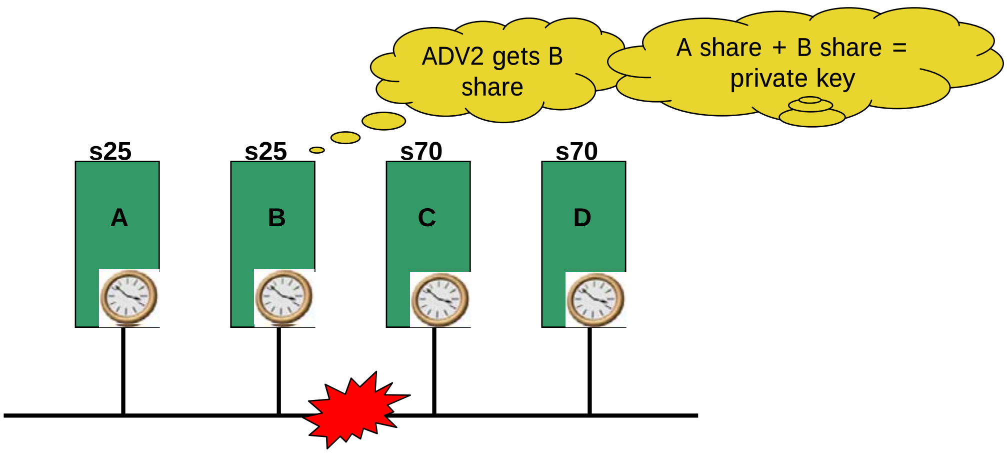
From Theory to Practice (6)

- Example with n=4, f=1



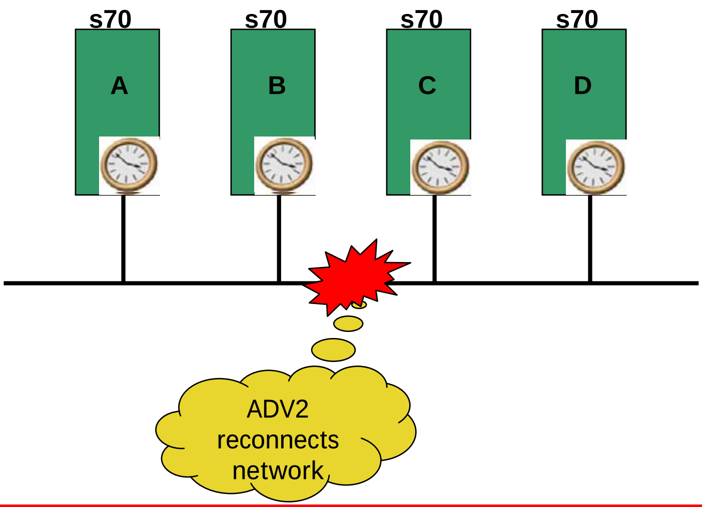
From Theory to Practice (6)

- Example with n=4, f=1



From Theory to Practice (6)

- Example with n=4, f=1

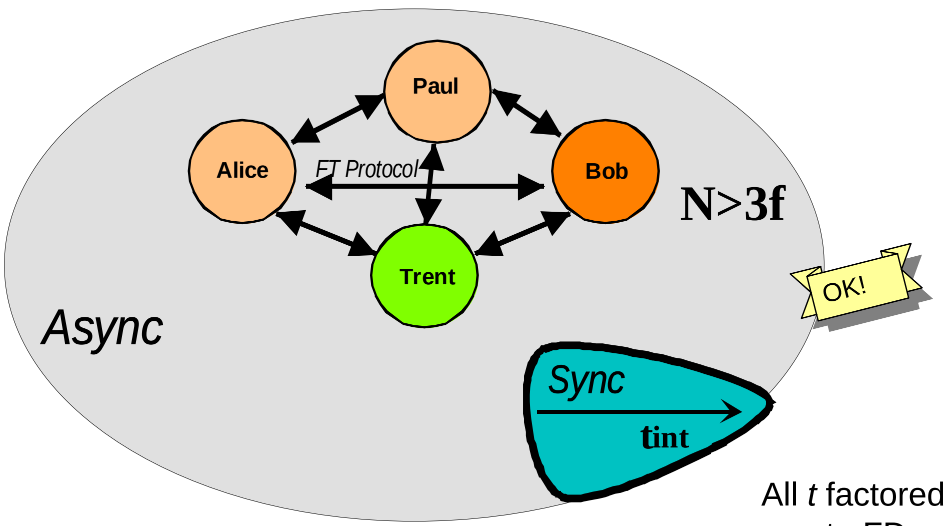


sXX: share version

The case for hybrid dis sys models

- in an asynchronous system, all timing assumptions must be encapsulated

Classical Model - Correct FT Async system



All *t* factored out
e.g. to FD oracles

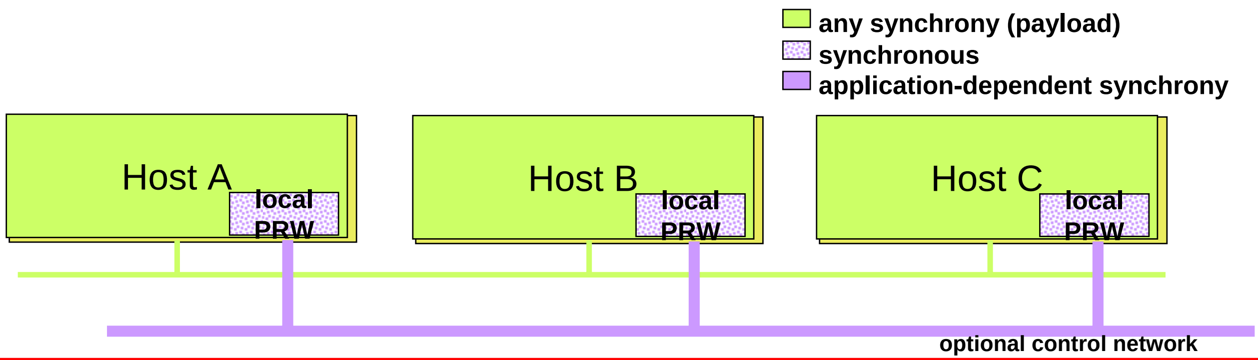
Findings

- Current state-of-the-art with homogenous models does not allow to construct exhaustion-safe distributed systems, specially in face of arbitrary/malicious faults:
  - Sync systems are vulnerable:
    - timing failures.
  - Async systems are vulnerable:
    - max number of faults + unbounded execution time.
  - Async systems with async proactive recovery are vulnerable:
    - max number of faults + unbounded rejuvenation period.

Proactive Resilience

Proactive Recovery in Wormhole (hybrid) models

- Using proactive recovery
  - define the number of faults between rejuvenations
  - compute rejuvenation period
  - execute recovery:
    - timely triggered
    - executed in bounded time



3

Models of Resilient Systems

Intrusion Tolerance

strategies

Failure assumptions in presence of intrusions

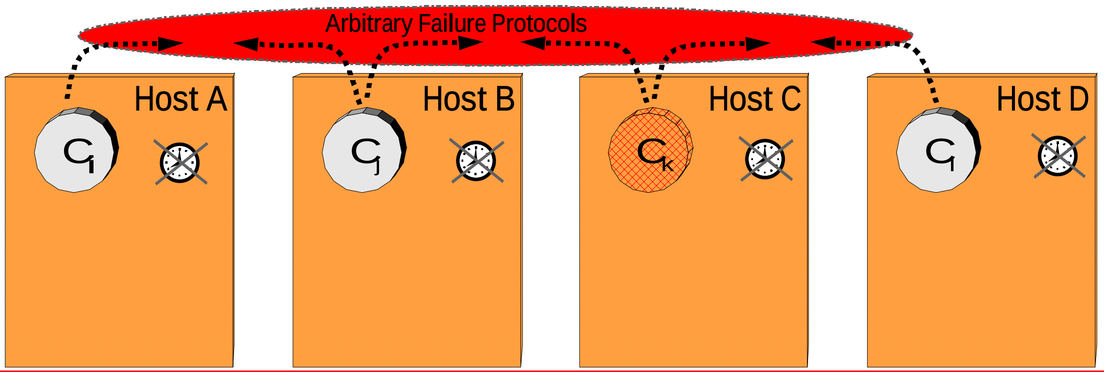
- Basic types of failure assumptions:
  - Controlled failures : assume qualitative and quantitative restrictions on failures, hard to specify for malicious faults
  - Arbitrary failures : unrestricted failures, limited only to the "possible" failures a component might exhibit, and the underlying model (e.g. synchronism)
- Fail-controlled vs. fail-arbitrary models in face of intrusions
  - FC have a coverage problem, but are simple and efficient
  - FA are normally inefficient, but safe
- What are *malicious* failures?
  - There is an adversarial attitude and an intention to harm
  - How do we model the mind and power of the attacker?

Modelling malicious failures

- Failures are no longer independent
  - Human attackers are the "common-mode" link
  - Triggering simultaneous attacks
  - Exploiting common vulnerabilities
  - Performing collusion through distributed protocols
- Failures become more severe
  - The worst possible behaviour: inconsistent output, at wrong times, forged, etc.
  - The greatest possible magnitude: patterns of occurrence no longer stochastic, only limited by attacker power
- Fault models become less representative
  - Maliciously induced failures defy qualitative (modes) and quantitative (stochastics) models for fault distribution

Asynchronous Fail-uncontrolled strategy

- Time-free
- Arbitrary failure environment
- Arbitrary failure protocols
- Used e.g. with: probabilistic Byzantine-agreement or consensus protocols
- Impossibility results for deterministic protocols, and for any timed operation

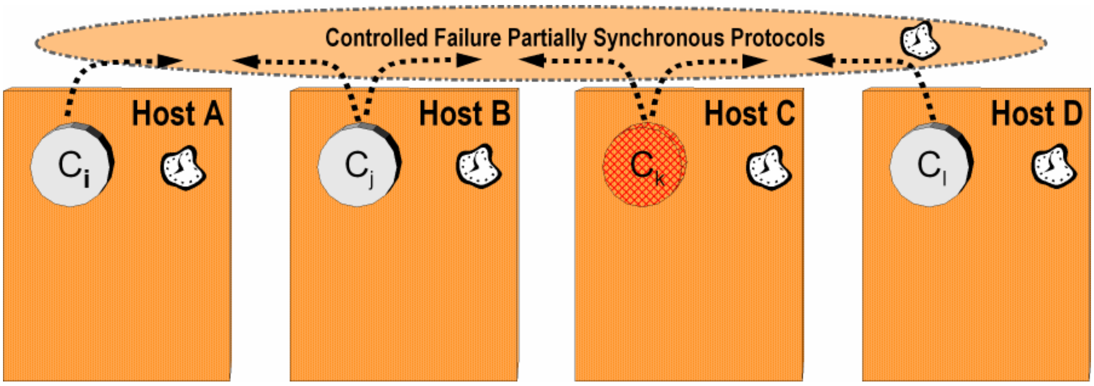


Arbitrary failure assumptions

- operations of very high value and/or criticality:
  - financial transactions of very high value (contracts,credencials)
  - critical control operations in infrastructures
  - whenever failure due to assumptions violation can't be incurred
  - AND, lack of performance and functionality can be accepted
- coverage of assumptions:
  - maximal, since little is assumed
- arbitrary-failure resilient building blocks
  - e.g. Byzantine agreement and consensus protocols
  - no assumptions on existence of fail-controlled components
  - impossibility of deterministic behaviour
  - time-free approach, impossibility of any timed operation

Partially-synchronous Fail-controlled strategy

- Timed, partially synchronous
- Non-Arbitrary failure environment and protocols
- Used e.g. with: classical reliable multicast and atomic broadcast
- Problem of coverage of assumptions



Recapitulating

- If you want efficient/performant solutions to F/T
  - assume controlled failure modes (omissive, fail-silent, etc.)
- If you want to build timely services (even soft R/T)
  - assume synchronous models, or at least partially sync
- Some security-related systems take this approach
  - partial synchronous environment
  - well-behaved (e.g. fortress) hosts
  - moderate level of threat in network
- They work, but only to the coverage of the assumptions
  - which must be substantiated
  - else we fall in the "*well-behaved hacker*" *syndrome*:
    - *``Hello, I'll be your hacker today, here is the list of what I promise not to do.''*
    - *``Oh thank you! By the way, here are a few additional attacks we would also like you not to attempt.''*

Where do we go from here?

- arbitrary failures / asynchrony thread
  - are safe, but normally inefficient
  - FLP: no deterministic solution of hard problems (e.g. ABCAST, consensus, BA)
  - does not solve timed problems (e.g., SCADA, CCC, e-com)
- controlled failures / synchrony thread
  - hard to specify for malicious faults and that brings a coverage problem
  - susceptible to attacks on timing assumptions
  - difficulty of implementation of sync. even in benign settings

Taking detours...

- OBJECTIVE:
  - solve most non-timed problems with highest possible coverage
- tone down determinism (e.g., randomisation)
- tone down liveness expectations (e.g., indulgence)
- use weaker semantics (e.g., thresholds, quorums)
- tone down allowed fault severity (e.g., hybrid faults)
- tone down asynchrony (e.g., parsync protocols, FDs)
- OBJECTIVE:
  - solve timed problems with highest possible coverage
- tone down asynchrony (e.g., sync/parsync protocols)

Take time/synchrony facet

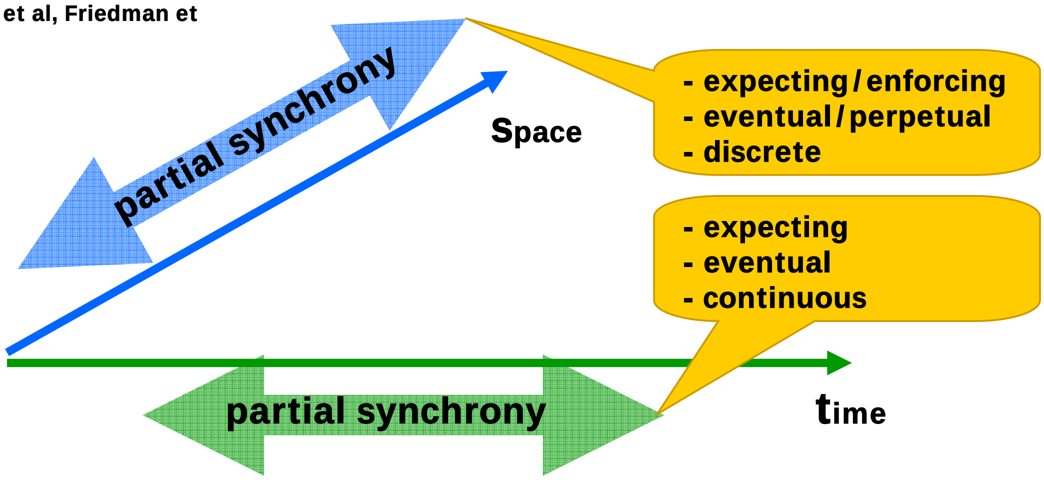
- OBSERVATION [Veríssimo and Casimiro. The Timely Computing Base model and architecture. DI/FCUL TR-99-2, IEEE TOCS 2002]:

  synchronism is not an invariant property of systems

- degree of synchronism varies in the time dimension:
  - during the timeline of their execution, systems become faster or slower, actions have greater or smaller bounds
- it also varies with the part of the system being considered, that is, in the space dimension:
  - some components are more predictable and/or faster than others, actions performed in or amongst the former have better defined and/or smaller bounds

Take time/synchrony facet

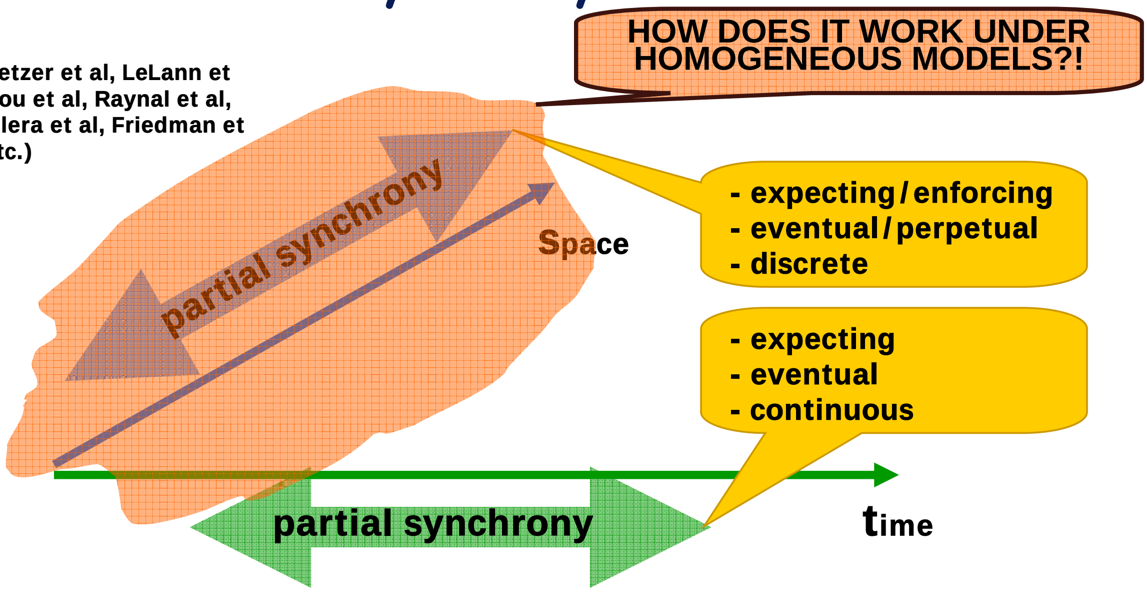
(Verissimo et al, Fetzer et al, LeLann et al, Castro et al, Zhou et al, Raynal et al, Macêdo et al, Aguilera et al, Friedman et al, Baldoni et al, etc.)



(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)

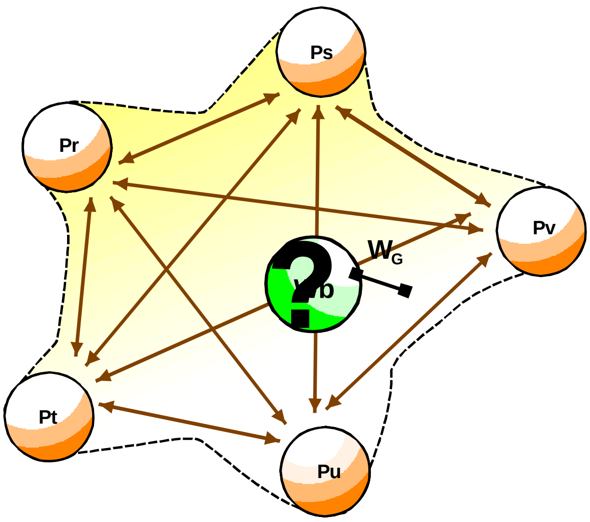
Take time/synchrony facet

(Verissimo et al, Fetzer et al, LeLann et al, Castro et al, Zhou et al, Raynal et al, Macêdo et al, Aguilera et al, Friedman et al, Baldoni et al, etc.)



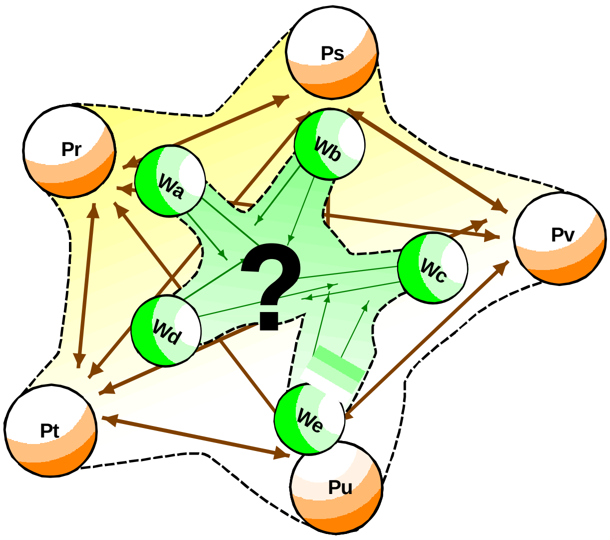
(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)

Homogeneous distr. sys. models



synchronous/secure   asynchronous/insecure

Homogeneous distr. sys. models



synchronous/secure asynchronous/insecure

Advanced modelling concepts

for IntTol systems

Advanced models for IntTol systems

- Recursive building of trust and trustworthiness
  - Trusted-trustworthy systems out of non-trustworthy components
- System models of hybrid trustworthiness
  - Trusted-trustworthy systems out of non-trustworthy AND trustworthy components

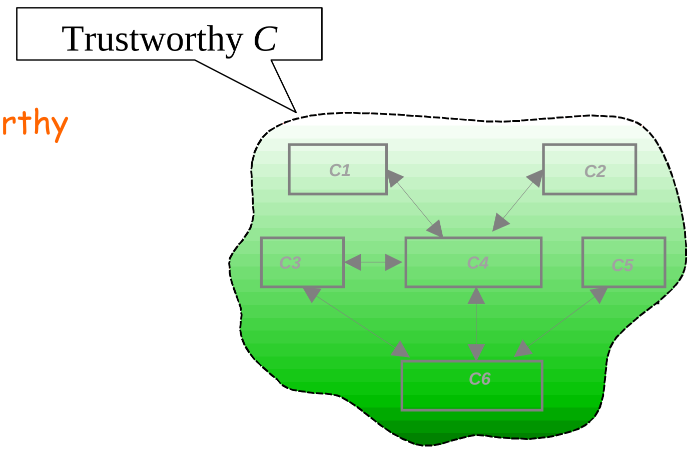
Advanced models for IntTol systems

- Intrusion-aware composite fault & intrusion models
  - the competitive edge over the hacker
  - AVI: attack-vulnerability-intrusion fault model
- Combined use of prevention and tolerance
  - malicious failure universe reduction
  - attack prevention, vulnerability prevention, vulnerability removal, in system architecture subsets and/or functional domains subsets
- Architecturally hybrid failure assumptions
  - different failure modes for distinct components
  - reduce complexity and increase performance, maintaining coverage
- Quantifiable assumption coverage
  - fault forecasting (on AVI)

Recursive building of trust & trustworthiness

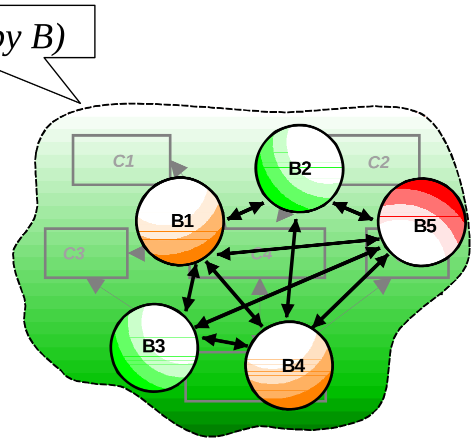
Building trustworthiness

- Subsystem C designed to be trustworthy
  - By construction



Building trust

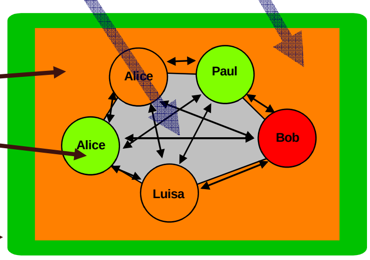
- Subsystem C designed to be trustworthy
  - By construction
- Subsystem C becomes *B*'s environment
  - Properties of C are assumed by B
- *B* trusts C
  - a trusted-trustworthy subsystem



On coverage and separation of concerns

- predicate P holds with a coverage Pr
  - we say that we are confident that P has a probability Pr of holding
- environmental assumption coverage (Pr_e)
  - set of assumptions (H) about the environment where system will run
  - $Pr_e = Pr(H \mid f)$ *f- any fault*
- operational assumption coverage (Pr_o)
  - the assumptions about how the system/algorithm/mechanism proper (A) will run, under a given set of environmental assumptions
  - $Pr_o = Pr(A \mid H)$

$Pr(A) = Pro \times Pre = Pr(A \mid H) \times Pr(H \mid f)$



System models of hybrid trustworthiness

Intrusion tolerance with hybrid failure assumptions

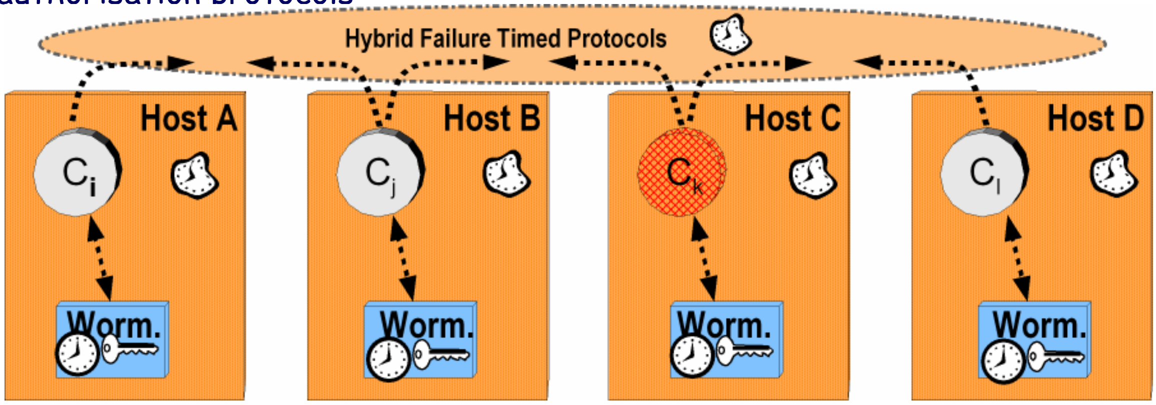
- How to achieve coverage of controlled failure assumptions, given unpredictability of attacks and elusiveness of vulnerabilities?
  - E.g. considering that not everything is intruded
- Hybrid failure assumptions:
  - the presence and severity of vulnerabilities, attacks and intrusions varies
- Classic hybrid fault models [Meyer, Pradhan, et al]
  - flat, use stochastic foundation to explain different behavior from a collection of components of same type (i.e. *k* crash and *w* byzantine in vector of values)
- Useless or at least risky in malicious environments
  - lack of substance: intentional player defrauds these assumptions

Hybrid failure assumptions considered useful

- Architectural hybridisation
  - the presence and severity of vulnerabilities, attacks and intrusions varies from component to component, i.e., different assumptions for distinct component subsets, possibly different
  - behaviour enforced by construction: trustworthiness
  - fail-controlled components or subsystems with justified coverage (trustworthy), used in the construction of fault-tolerant protocols under hybrid failure assumptions
- Using trusted-trustworthy components or subsystems:
  - black boxes with benign behavior, omissive or weak fail-silent type
  - different capabilities (e.g. synchronous or not; local or distributed), can exist at different levels of abstraction

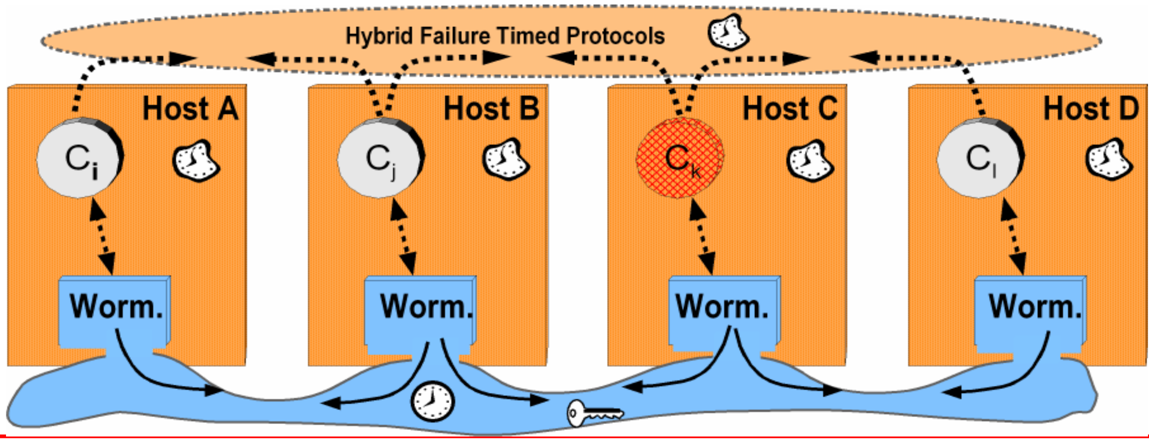
Fail-controlled IntTol system models
with Local Trusted Components

- Trustworthy subsystem (also called Wormhole) - e.g. smart or Java card; appliance board
- Secure, and time-free or timed (as in figure)
- Arbitrary failure environment + Local Wormhole
- Hybrid failure protocols
- Example usage: FT distributed data dissemination with authentication and authorisation protocols



Fail-controlled IntTol system models
with Distributed Trusted Components

- Distributed Trustworthy subsystem (distr. Wormhole) - e.g. appliance boards interconnected by dedicated network
- Secure, and time-free or timed (as in figure)
- Arbitrary failure environment + Distributed Wormhole
- Hybrid failure protocols
- Example: FT transac. prots requiring timing constraints (e.g. SCADA, DCS)

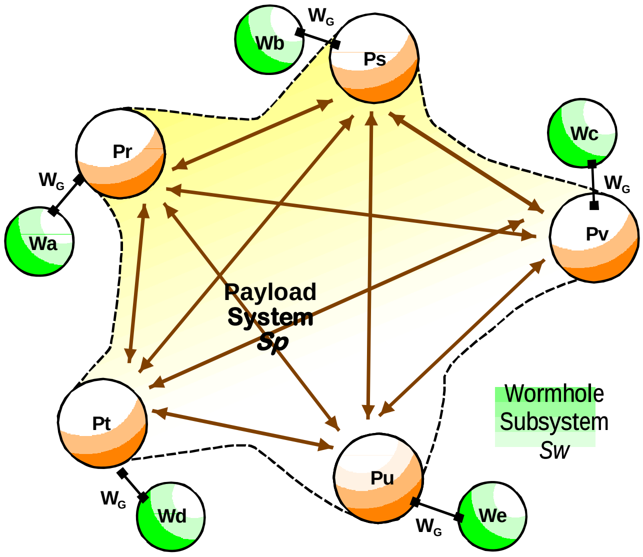


Design for Resilience

Architecurally hybrid distributed systems models

© 2002-08 Paulo Veríssimo - All rights reserved, no unauthorized reproduction in any form

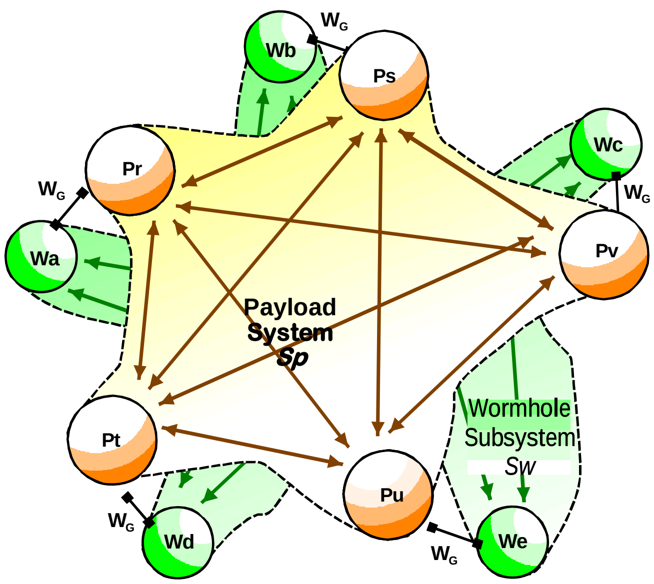
Architect.hybrid distr. sys. models



Any-synchrony / security system P   Any-synchrony / security system W

Design for Resilience

Architect.hybrid distr. sys. models



Any-synchrony / security system P   Any-synchrony / security system W

Shortcuts vs. detours

- Rendering the solution simpler
  (without changing the problem!)

- Architectural hybridization
- Wormholes model

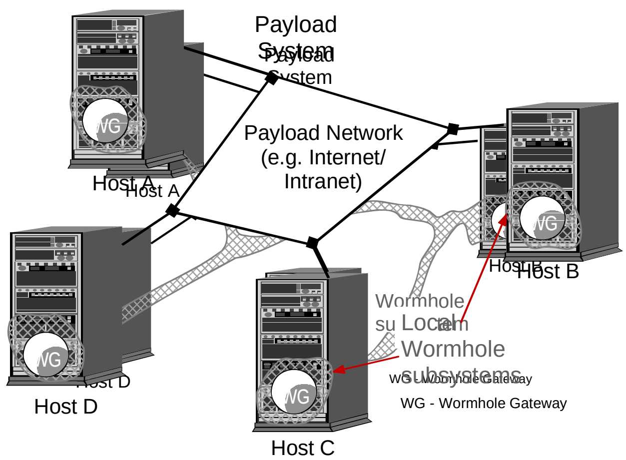
In

Paulo Veríssimo, [Travelling through Wormholes: a new look at Distributed Systems Models], SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), vol. 37, no. 1, (Whole Number 138), 2006.

Paulo Veríssimo, [Uncertainty and Predictability: Can they be reconciled?], Future Directions in Distributed Computing, pp. 108-113, Springer Verlag LNCS 2584, May, 2003

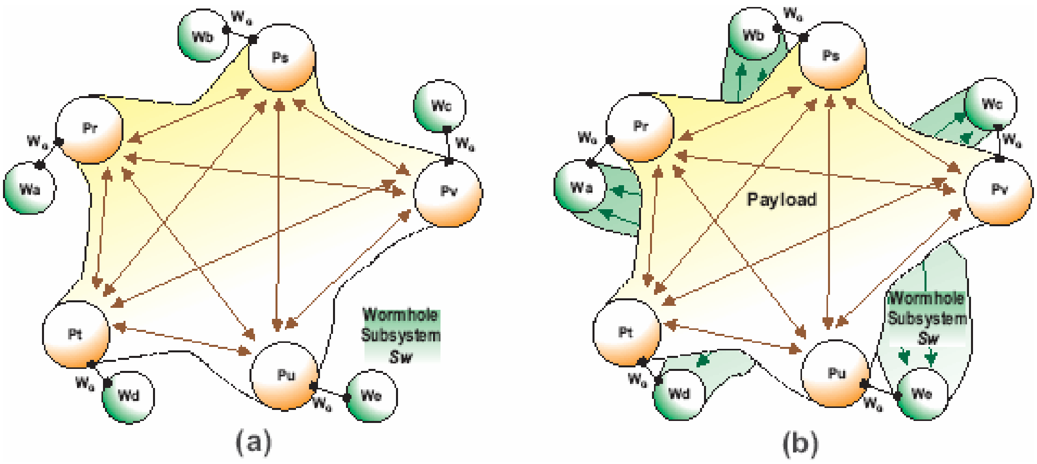
Wormholes

- New design philosophy for distributed systems:
- constructs with privileged properties which endow systems with the capability of evading the uncertainty of the environment (``taking a shortcut'') for certain crucial steps of their operation, in order to achieve the required "hard properties" (predictability)



Theoretical underpinnings

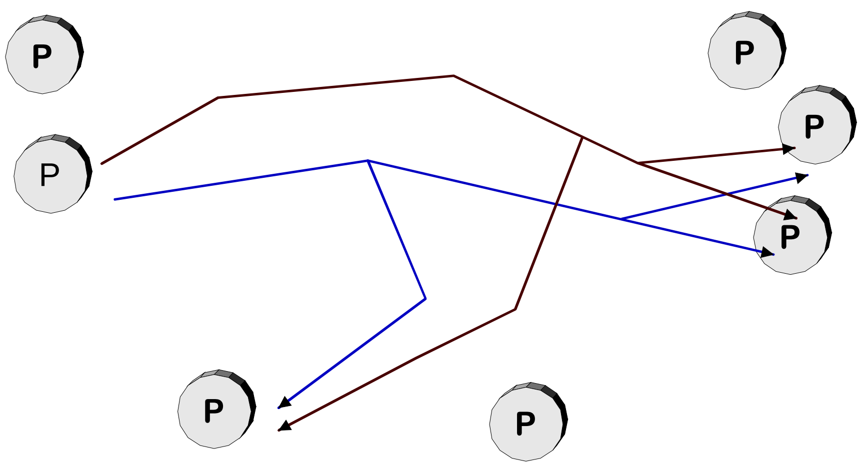
- A generic hybrid distributed systems model, or Wormholes Model



Design for Resilience

Theoretical underpinnings

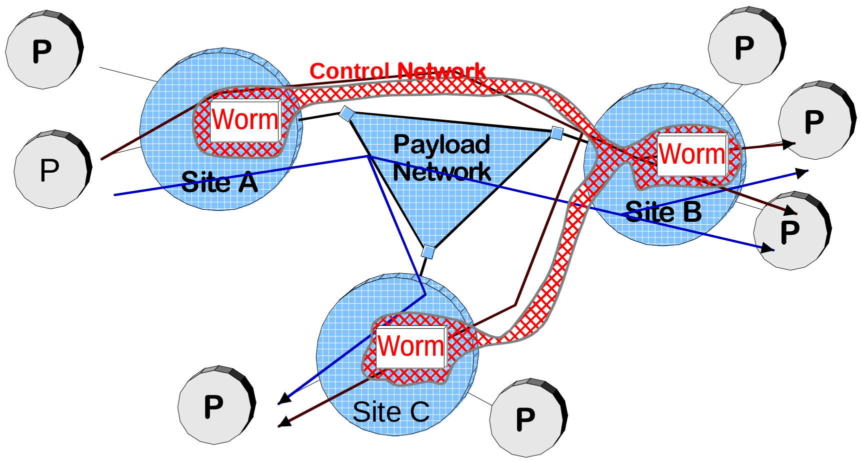
- Processes and links in Wormholes models



© 2002-08 Paulo Veríssimo - All rights reserved, no unauthorized reproduction in any form

Theoretical underpinnings

- Architecture imprinting in Wormholes models



An example Wormhole:
Trusted Timely Computing Base (TTCB)

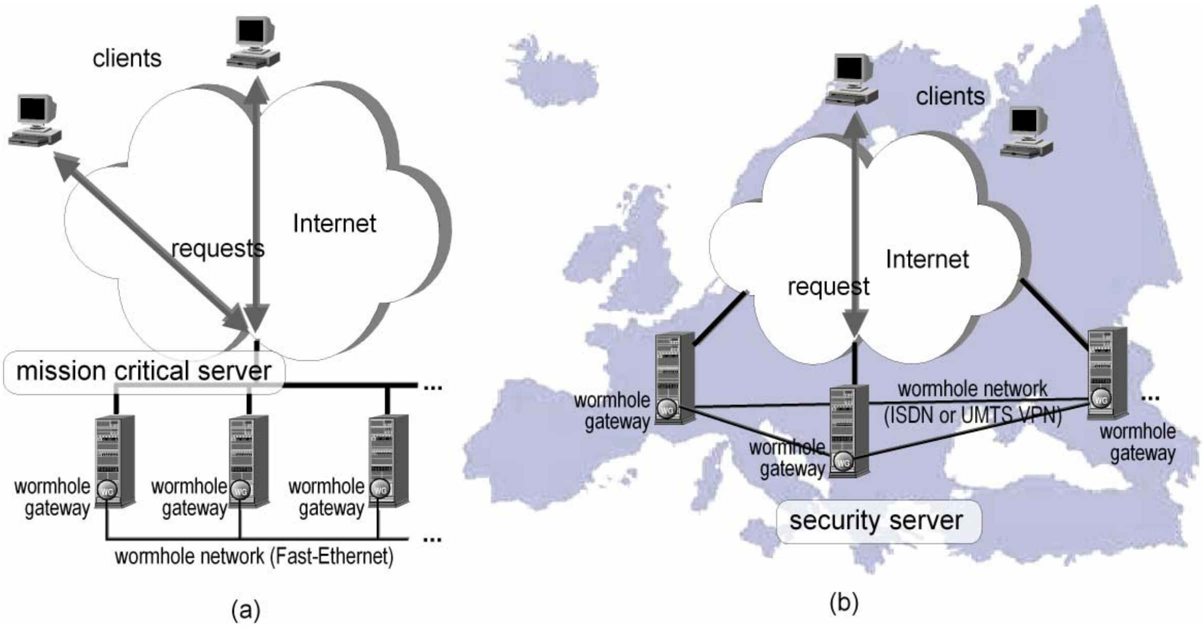
- Properties:
  - trusted and timely execution; trusted timing failure detection
  - secure (can only fail by crashing)
  - real-time (capable of timely behavior)
  - correct processes can interact securely with the TTCB
- TTCB can be seen as a distributed security kernel that provides a minimal set of trusted and timely services to assist the execution of fault/intrusion-tolerant algorithms, such as :
  - provides a trusted environment for crucial steps
  - local authentication
  - agreement on a fixed sized block of data (TBA)
  - globally meaningful timestamps

– Can be built (there is a COTS-based prototype)

Correia, Veríssimo, and Neves. The Design of a COTS Real-Time Distributed Security Kernel. European Dependable Computing Conf., *EDCC-4*, October 2002

Wormholes model in action

Example of deployment of systems with wormholes

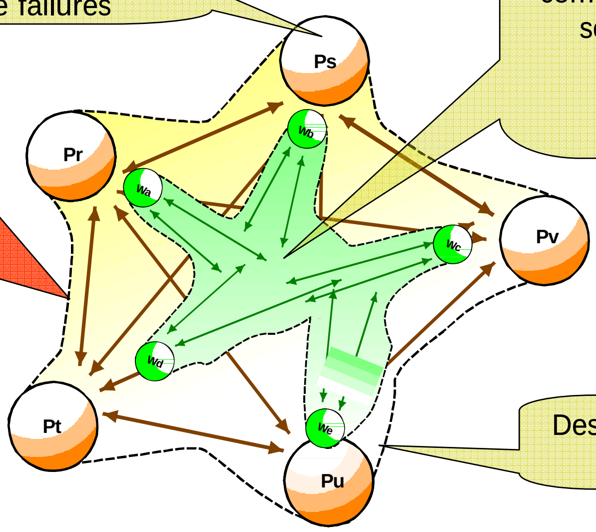


Designing algorithms with wormholes (aka hybrid distributed systems models)

Assume basic system P model, e.g. asynch. and Byzantine failures

Postulate existence of components (W) on a different set of assumptions, e.g.:
- failure detector oracle
- set of fast(er) or synch. channels

- Proof correct conditional to truthfulness of assumptions.
- What if assumptions cannot be substantiated?
- I.e. they do not represent physical reality?

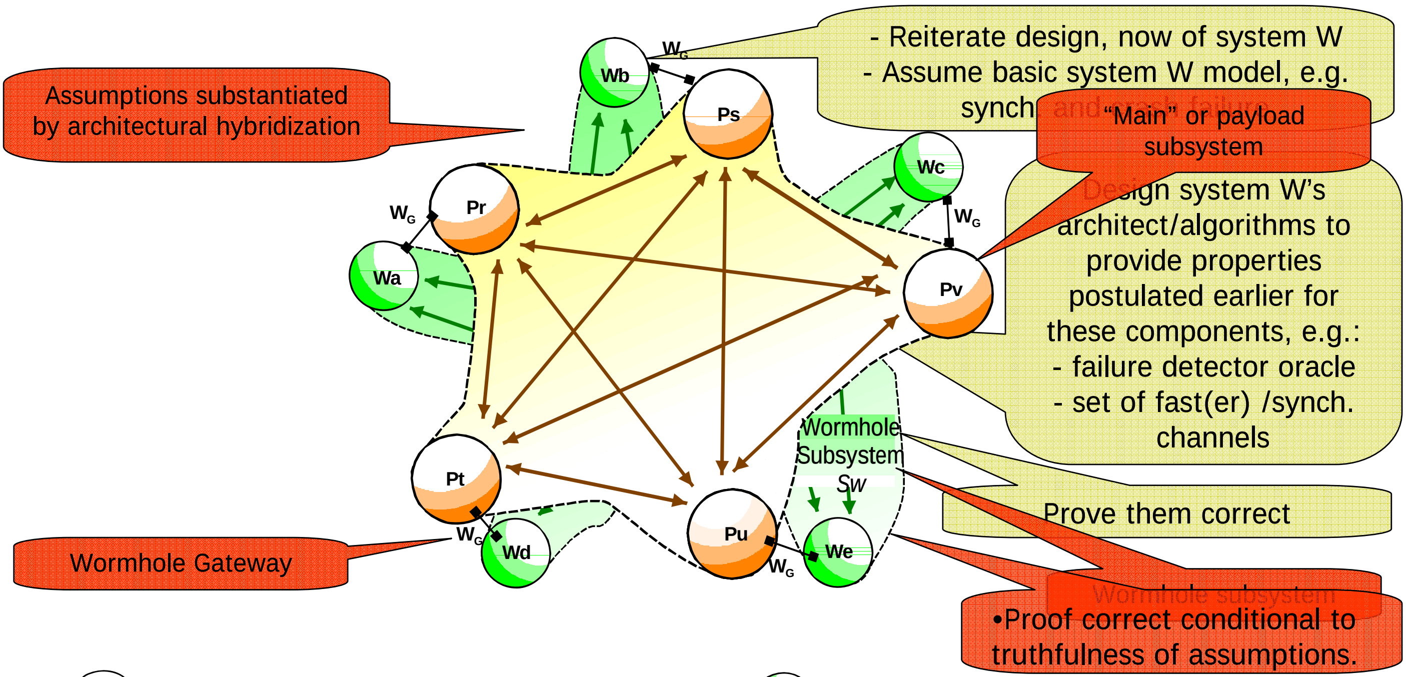


Design your P algorithms and prove them correct

Any-synchrony / security system P

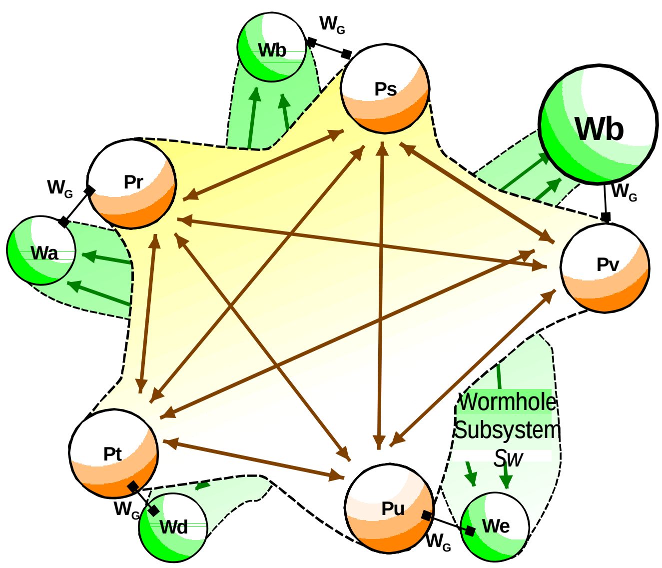
Any-synchrony / security system W

Designing algorithms with wormholes (aka hybrid distributed systems models)



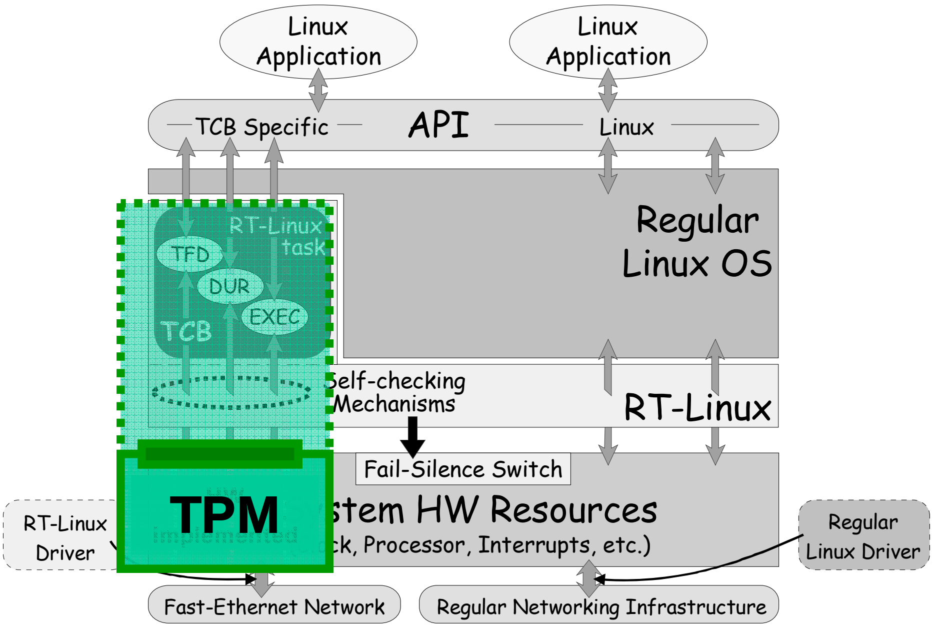
Any-synchrony/security system P **Any-synchrony/security system W**

Proof-of-concept systems with wormholes



Any-synchrony/security system P **Any-synchrony/security system W**

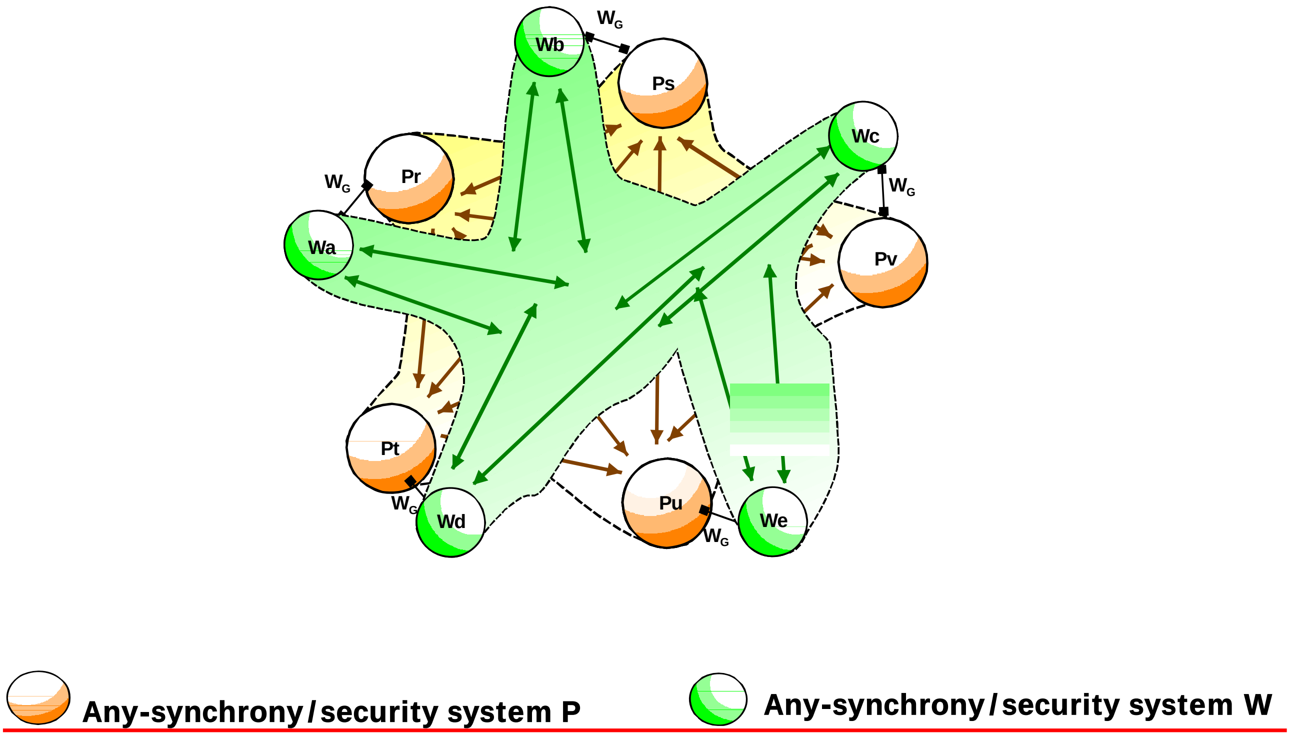
Proof-of-concept: COTS-based TCB Reference Architecture



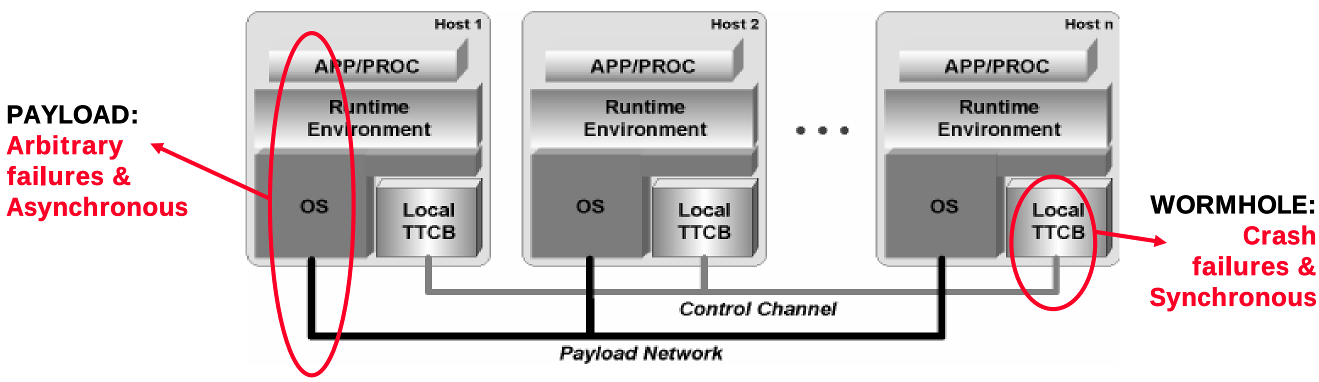
Example Hardware-based Wormholes

- Connectivity:
  - Wireless WiFI, Bluetooth
  - Wired RS-232, USB2, Ethernet

Proof-of-concept systems with wormholes



Proof-of-concept: Distr. crash failure synch. wormhole

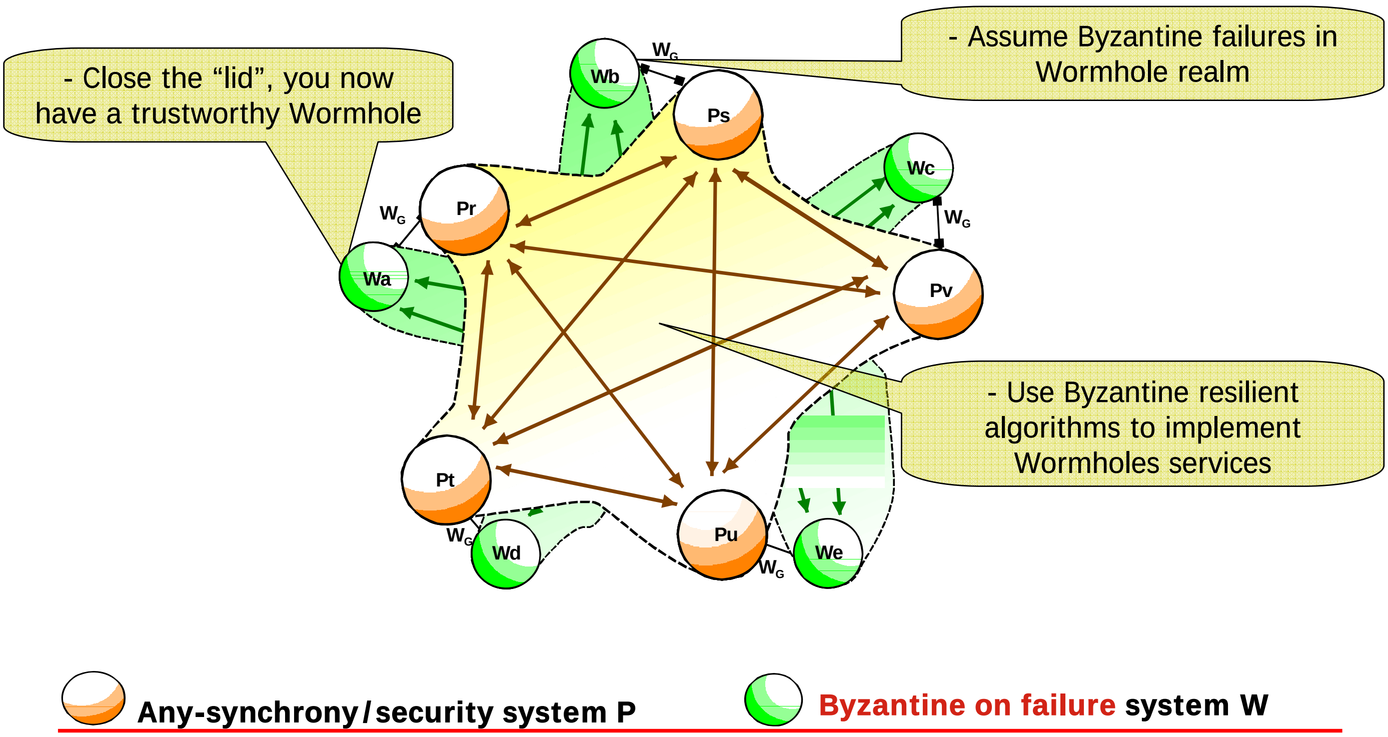


- TTCB is a distributed real-time and security kernel that provides a minimal set of trusted and timely services, such as
  - failure detection
  - local authentication
  - agreement on a fixed sized block of data (TBA)
  - trustworthy global timestamps and random numbers

Weaker wormholes

- Wormholes can be *any* distributed subsystem/component that follows different assumptions from "main" (payload) system:
  - watchdog
  - crypto chip
  - sync or parSync set of channels
  - timely execution monitor
- There can be more than one wormhole subsystem
- Wormhole subsystems can be constructed as fault or intrusion-tolerant subsystems

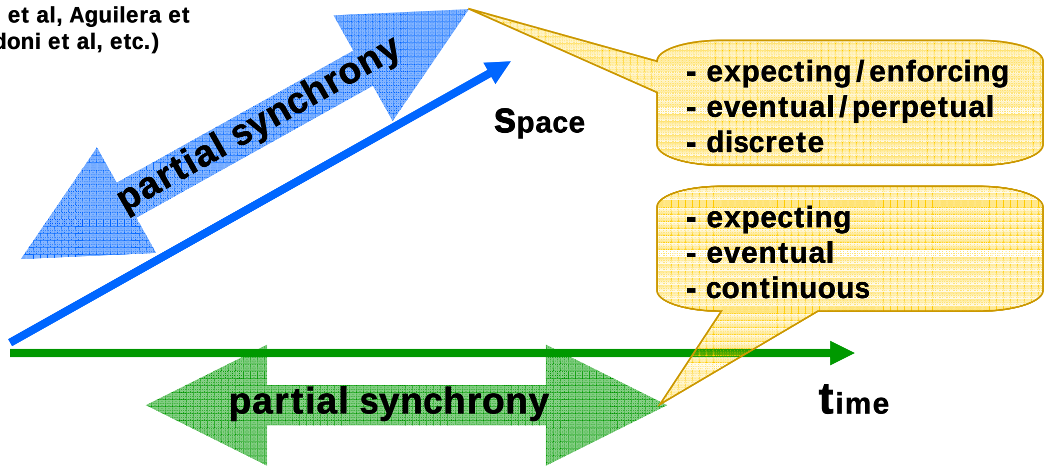
Proof-of-concept systems with wormholes
Fault/Intrusion-tolerant wormholes



Hybrid models/architectures more complex than homogenous, why use them?

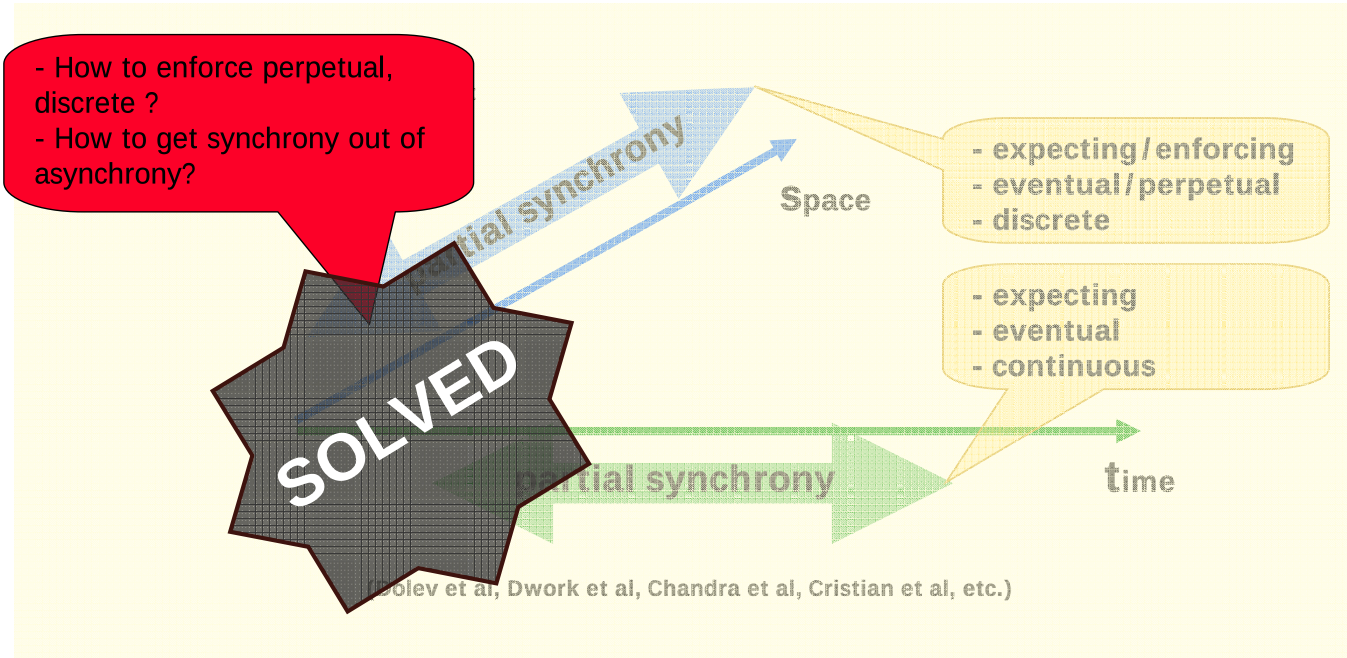
Take time/synchrony facet

(Verissimo et al, Fetzer et al, LeLann et al, Castro et al, Raynal et al, Aguilera et al, Friedman et al, Baldoni et al, etc.)



(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)

Take time/synchrony facet

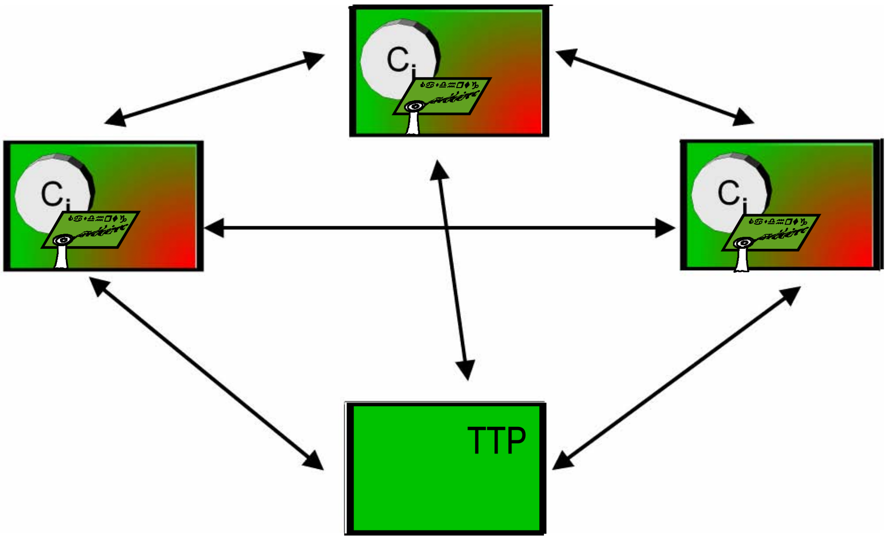


Review of

Strategies for construction

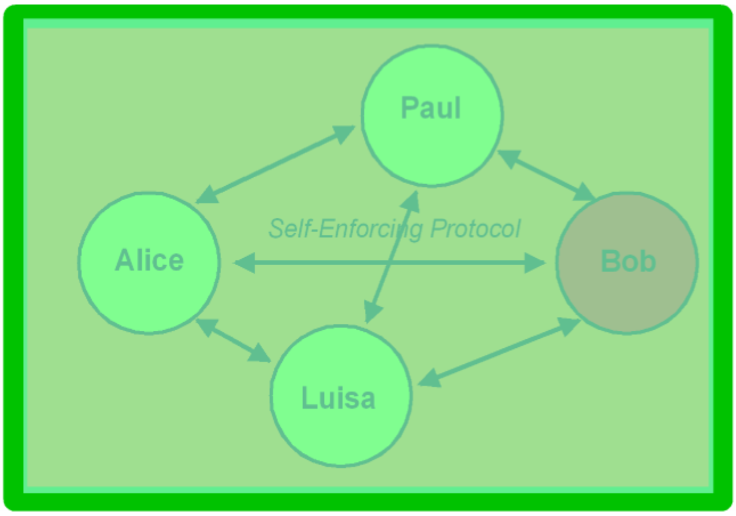
of IntTol subsystems

Recursive use of F. Prevention and F.Tolerance



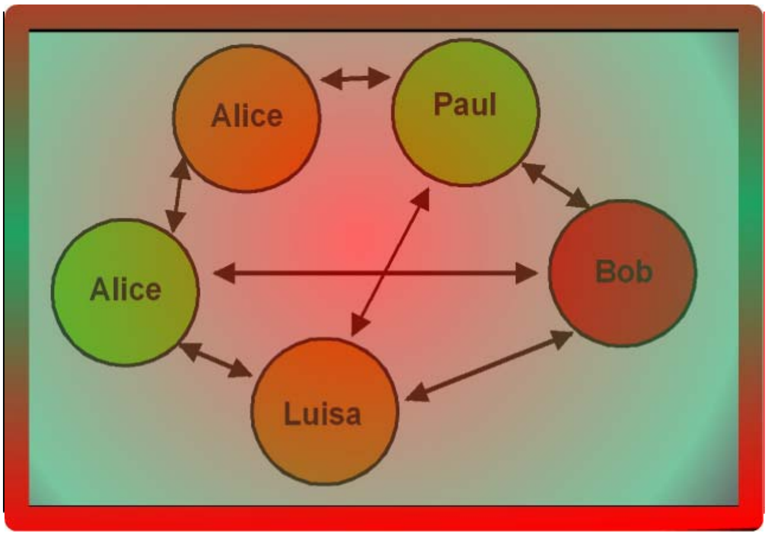
- The TTP protocol revisited
- Work at subsystem level to achieve justifiable behaviour
- Architectural hybridation w.r.t. failure assumptions

Strategies for construction of IT subsystems



- Arbitrary model – no assumptions
- High coverage – very little to "cover"

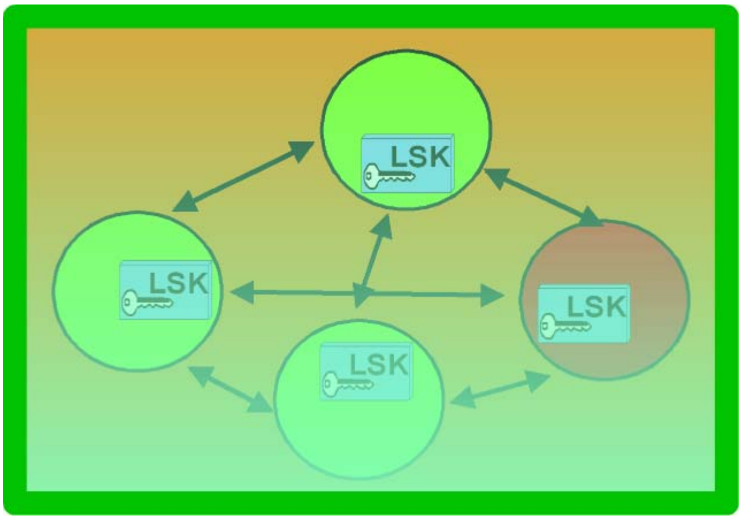
Strategies for construction of IT subsystems



- Fail-controlled model -- unjustified environment assumptions
- Fair coverage – no enforcement

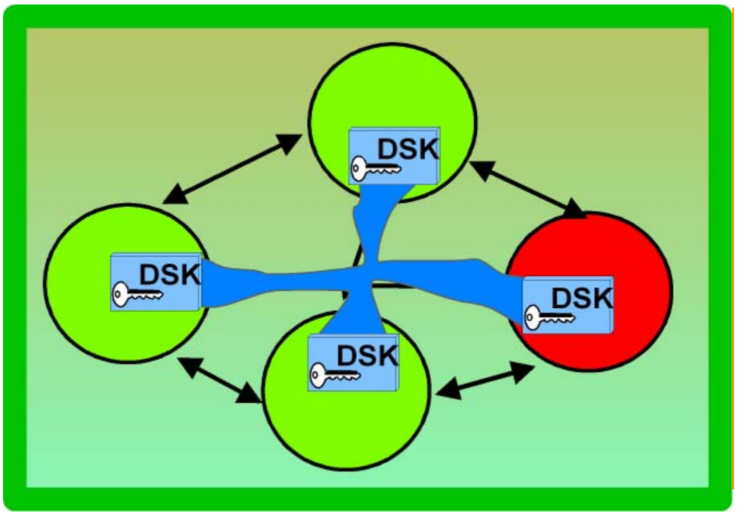
Design for Resilience

Strategies for construction of IT subsystems



- Fail-controlled model – little environment assumptions; justified component assumptions
- High coverage – enforcement by Local Trusted Comp.

Strategies for construction of IT subsystems



- Fail-controlled model – little environment assumptions; justified component assumptions
- High coverage – enforcement by Distr. Trusted Comp.

Wormhole-Aware Byzantine Protocols

Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Fault Model

Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model, ***Miguel Correia, Lau Cheuk Lung, Nuno Ferreira Neves, Paulo Veríssimo.*** *Proc's of the 21st Symp. on Reliable Distributed Systems (SRDS'2002), Suita, Japan, October 2002*

Basic failure modes

- Processes can fail in a Byzantine way:
  - Crash, disobey the protocol, send contradictory messages, collude with other malicious processes,...
- Network:
  - Can corrupt packets (due to accidental faults)
  - An attacker can modify, delete, and introduce messages in the network

TTCB services

- The *reliable multicast* protocol uses only three *TTCB services:*
- Local authentication service
- Trusted block agreement
- Trusted absolute timestamping

Agreement Service

- A process makes two operations:
  - propose, decide
  - this works with “small” blocks of data
- *agreement* is defined by (elist, tstart, decision)
  - elist: list of processes involved
  - tstart: instant when the TTCB stops accepting proposals
  - decision = TTCB_TBA_RMULTICAST; returns:
    - value proposed by 1st process in elist
    - mask *proposed-ok:* processes that proposed the value decided

First phase

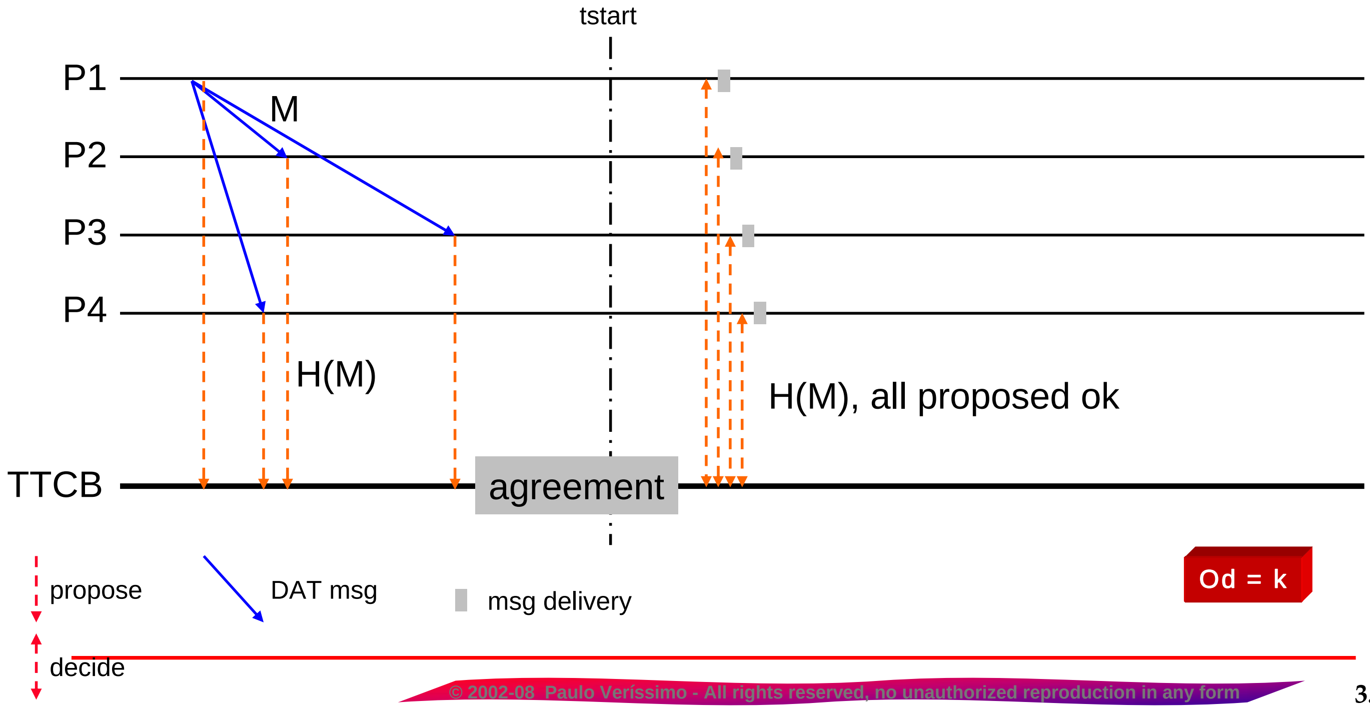
- The protocol terminates in the first phase if there are no faults or delays
- The sender:
  - sends a data message (DAT)
  - give the recipients a reliable **hash** of the message sent using the TTCB Agreement Service
- The TTCB Agreement Service acknowledges the processes that proposed the right hash
  - if all proposed the protocol terminates

BRM-M Sender and Recipient protocol

```
// ——— Phase 1 ———
if I am the sender then                        // SENDER process
    M := (DAT, my-eid, elist, TTCB_getTimestamp() + T1, data);
    multicast M to elist except sender; n-sends := 1;
else                                           // RECIPIENT processes
    read_blocking(M); n-sends := 0;
propose := TTCB_propose(M.elist, M.tstart,
    TTCB_TBA_RMULTICAST, H(M));
do decide:=TTCB_decide(propose.tag);
    while (decide.error≠TTCB_TBA_ENDED);
if (decide.proposed-ok contains all recipients) then deliver M; return;
// ——— Phase 2 ———
M-deliver := ⊥;
mac-vector := calculate macs of (ACK, my-eid, M.elist, M.tstart,
    decide.value);
M-ack := (ACK, my-eid, M.elist, M.tstart, mac-vector);
n-acks := 0; ack-set := eids in decide.proposed-ok;
t-resend := TTCB_getTimestamp();
do
    if (M.type = DAT) and (H(M) = decide.value) then
        M-deliver := M;
        ack-set := ack-set ∪ {my-eid};
        if (my-eid ∉ decide.proposed-ok) and (n-acks < Od+1) then
            multicast M-ack to elist except my-eid; n-acks := n-acks + 1;
    else if (M.type = ACK) and (M.mac-vector[my-eid] is ok) then
        ack-set := ack-set ∪ {M.sender};
    if (M-deliver ≠ ⊥) and (TTCB_getTimestamp() ≥ t- resend) then
        multicast M-deliver to elist except (sender and eids in ack- set);
        t-resend := t-resend + Tresend; n-sends := n-sends + 1;
    read_non_blocking(M); // sets M = ⊥ if no messages to be read
while (ack-set does not contain all recipients) and (n-sends < Od+1);
deliver(M-deliver);
```

Figure 2. BRM-M protocol.

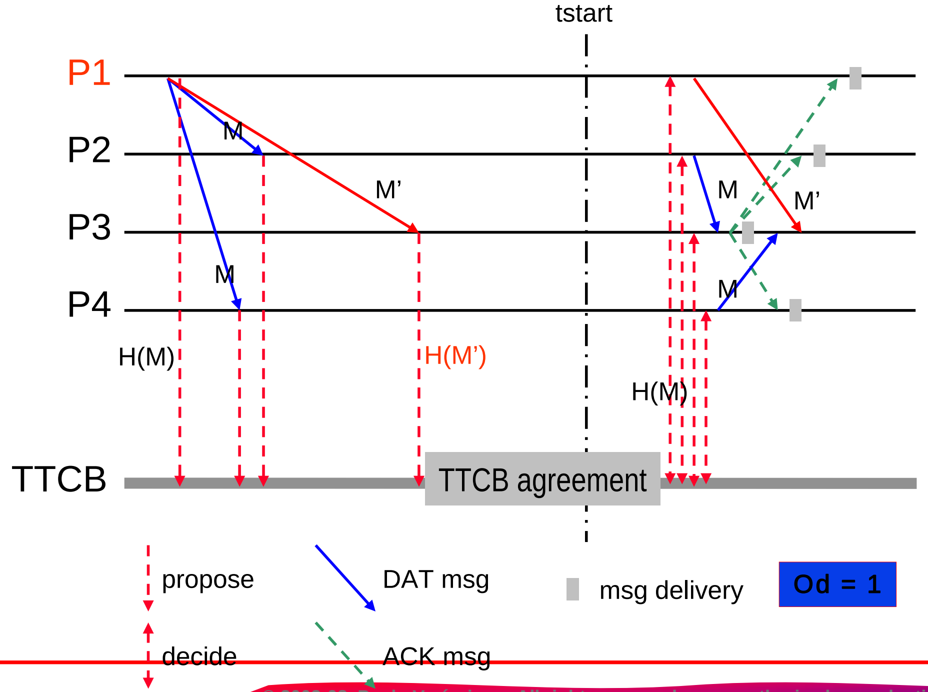
Example: best case (1st phase only)



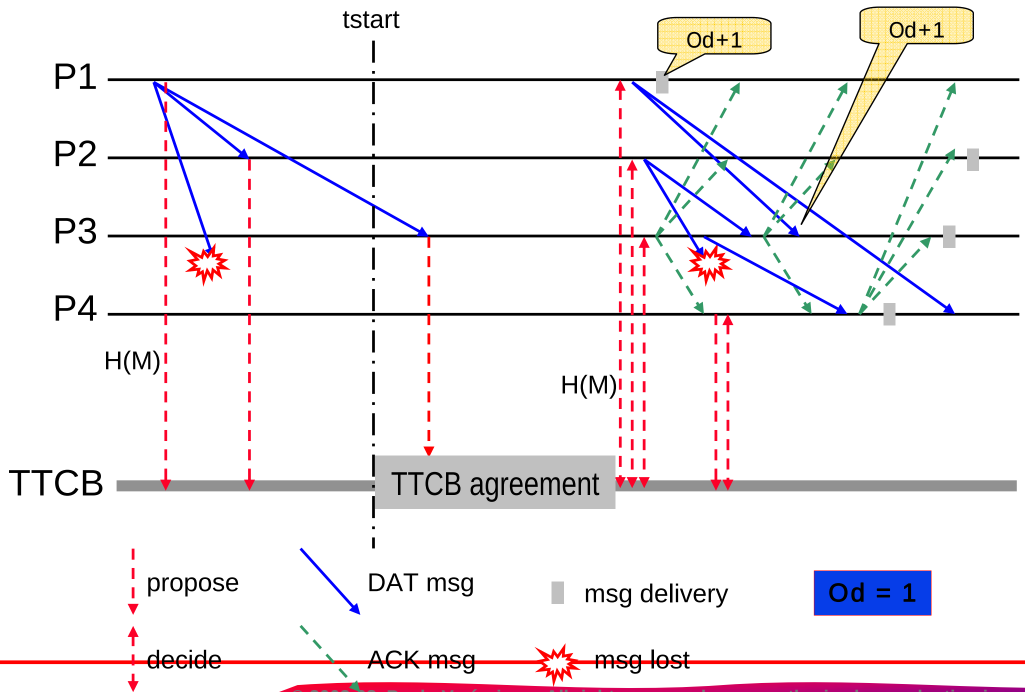
Second phase (II)

- Each process that has the message for which H(M) = value returned by the TTCB Agreement, resends M until:
  - All processes acknowledged:
    - Proposing on time for the TTCB Agreement; or
    - With an ACK
  - Or until it sent Od+1 times:
    - Processes that do not receive are failed

Example: malicious sender

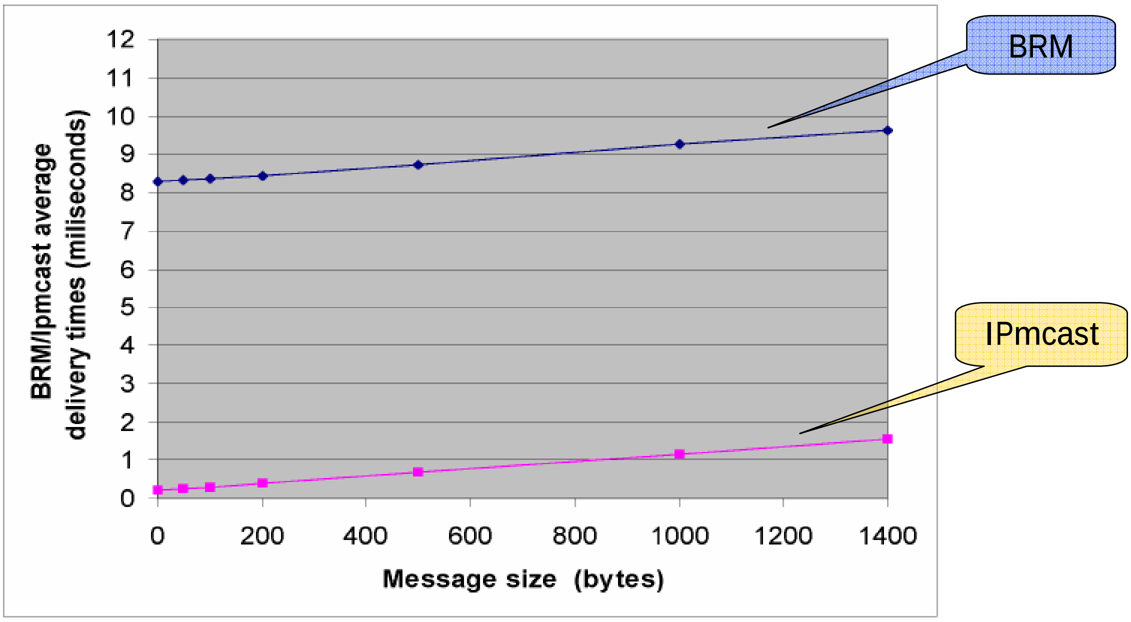


Example: message losses/delays



3. Protocol Performance

Measurements



Typical values in earlier works: ~50ms

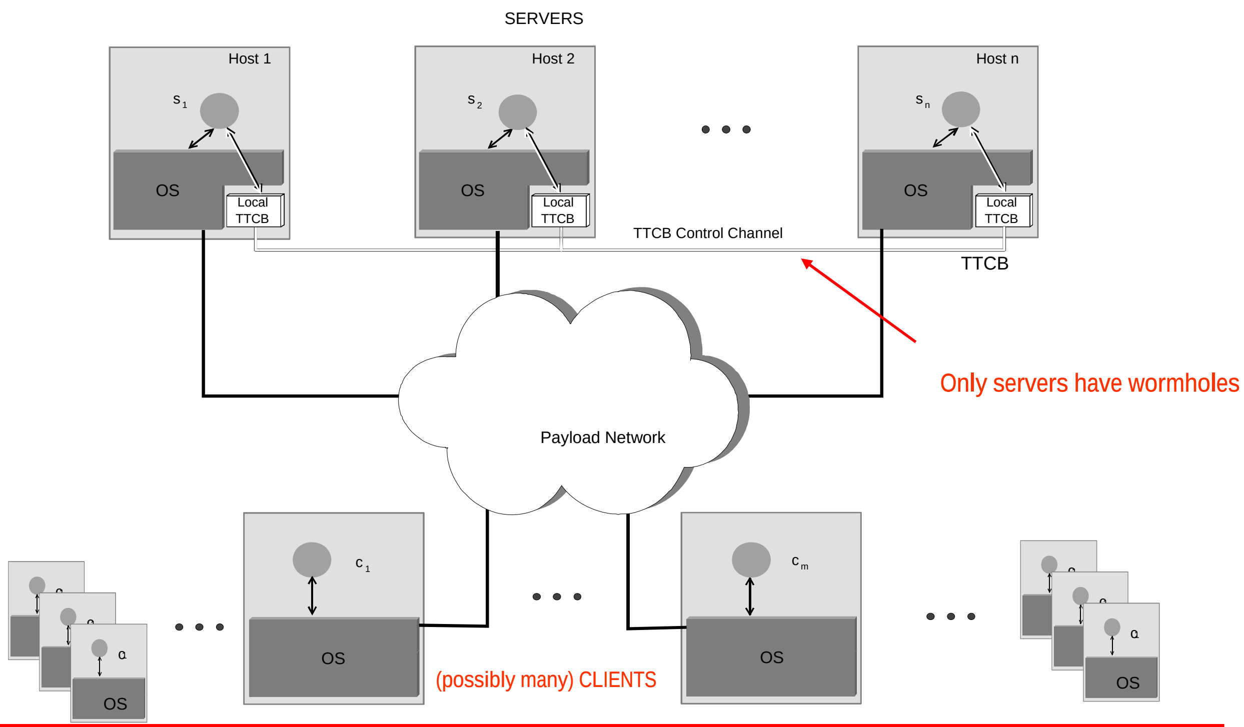
Achievements

- Reliable multicast with Byzantine faults requires:
  - asynchronous system: $n \geq 3f+1$ [Bracha&Toueg]
  - synchronous system: no limit ($n \geq f+2$) [Lamport et al.]
- We follow a wormhole-aware model:
  - payload is asynchronous and byzantine-on-failure
  - TTCB is synchronous and crash-on-failure
- We achieve:
  - $n \geq f+2$ without asymmetric crypto (signatures)
  - Efficiency: few phases, high performance

State machine replication on atomic multicast

How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems. ***Miguel Correia, Nuno Ferreira Neves, Paulo Veríssimo***. *In Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems. Florianopolis, Brasil, pages 174-183, October 2004.*

System architecture



Achievements

- First SMA service for practical byzantine distributed systems with resilience f out of 2f+1
  - Lower number of replicas reduces cost of hardware + cost of designing different replicas (for fault independence)
- Low time complexity
- Good performance since it does not resort to public key cryptography