Fault Tolerance in Distributed Systems

an introductory course

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

Course Structure

- Introduction to distributed systems: foundations, main paradigms and models
- Distributed fault tolerance foundations and paradigms
- Models and systems for distributed fault-tolerant computing
- Case study: making VP'63 distributed and dependable
- Introduction to the fundamental concepts of intrusion tolerance

Pointers to course material

- Material for *distributed systems* review, and material for *fault tolerance* concepts:
  - The book Distributed Systems For System Architects, Paulo Verissimo and Luís Rodrigues, 2001, Kluwer Academic Publishers. http://www.navigators.di.fc.ul.pt/dssa
  - Further reading for these topics is described in the book, at end of each chapter.
- Material for the newer *intrusion tolerance* domain, is available from the University of Lisbon web site:
  - Intrusion-Tolerant Architectures: Concepts and Design Verissimo, P., Neves, N., and Correia, M. In: Architecting Dependable Systems. Springer-Verlag LNCS 2677 (2003). Ext. version, Tech. Rep. DI/FCUL TR03-5, Dept. of Informatics, Univ. of Lisboa (2003). http://www.navigators.di.fc.ul.pt/it/index.htm

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 Intern. 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 Int. Confer. 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.

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. Verissimo, 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. Verissimo. Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model. In Proc. of Symp. of Reliable Distributed Systems, October 2002, Japan.
- M. Correia, P. Verissimo, 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 Procs of the 26th IEEE Symp. 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), LNCS, 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 Int'l Conf. on Dependable Sys. and Networks (DSN), Washington-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.
- 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 Sys. (S. Krakowiak and S. Shrivastava, eds.), vol.1752, 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.

Further Reading

- 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), Washington, USA, 2002.
- Matti A. Hiltunen, Richard D. Schlichting and Carlos A. Ugarte, Enhancing Survivability of Security Services Using Redundancy. In Procs. of the Int'T Conf. on Dependable Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.
- 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, Philadel.,USA.
- Gunjan Khanna, Mike Yu Cheng, Padma Varadharajan, Saurabh Bagchi, Miguel Correia, Paulo Verissimo. Automated Rule-Based Diagnosis Through A Distributed Monitor System. IEEE Trans. on Dependable and Secure Computing, vol.4, no.4, pp. 266-279, 2007.
- K. Kihlstrom, L. Moser, and P. Melliar-Smith, The SecureRing protocols for securing group communication," Proc. 31st Hawaii Int'l Conf. on Sys. Sci., 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.

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 Verissimo, 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. Verissimo, 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. Verissimo 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 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 Verissimo, 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 Verissimo, 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 Verissimo, 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. Verissimo, 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 Distributed Systems

What is a distributed system?

A computer network is not a distributed system

- *computer network*
  - infrastructure serving a set of computers interconnected through communication links of possibly diverse media and topology, and using a common set of communication protocols.
- *distributed system*
  - system composed of several computers which communicate through a *computer network*, hosting processes that use a common set of distributed protocols to assist the coherent execution of distributed activities.

Characteristics of Distributed Systems

- multiple computers
- interconnected by a network
- sharing state

- independent failures
- communication is unreliable
- has variable delays
- speed and bandwidth are moderate
- investment costs often lower than mainframes
- management costs are higher
- partial ordering of events only
- difficult to assess global state

Centralized versus Distributed Systems

- **Centralized** :
  - Accessibility
    - to resources and info
  - Homogeneity
    - of procedures and technologies
  - Manageability
    - single, domain
  - Consistency
    - global state
  - Security, due to:
    - threat reduction
- **Distributed** :
  - Scalability, helped by:
    - geographical scope
    - heterogeneity
    - modularity
  - Sharing:
    - of common resources and info
  - Reliability and availability, due to:
    - redundancy and replication
    - graceful degradation
    - failure independence
  - Security, due to:
    - vulnerability reduction
  - Low cost factor

Decentralization

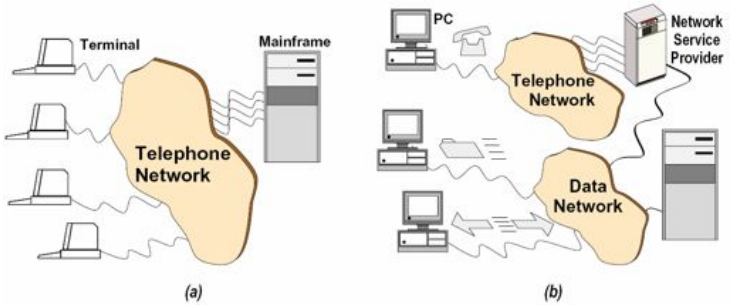
- Decentralization *is not* distribution
  - decentralization is local autonomy of means and procedures, based on local control points that contribute to the goals of a wider structure
- distribution adapts computing infrastructure to a decentralized model of activity
- strategy for decentralization?
  - decentralize control, integrate and coordinate activities
  - supply coherent system view to all loci of control
    - *common knowledge*
  - secure decentralized coordination of actions
    - *distributed algorithmic*

Distributed Systems Evolution

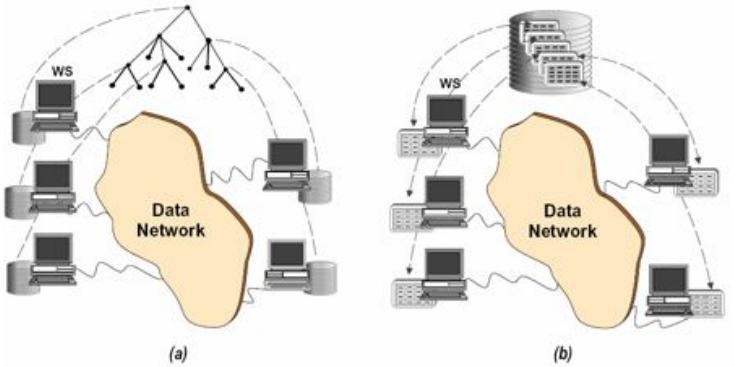
ARCHITECTURES

Remote Access Architectures:

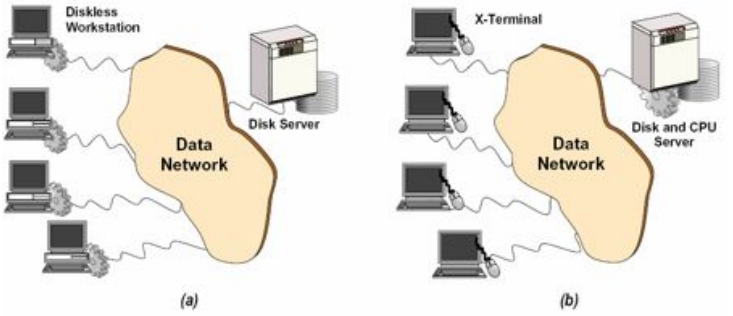
(a) Plain Telephone Line; (b) Data Network



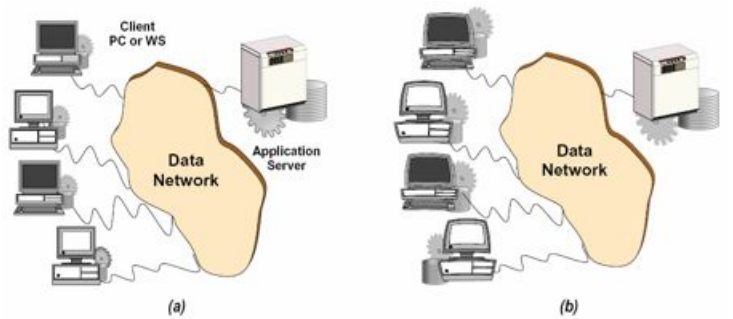
Distributed File and Memory Architect.



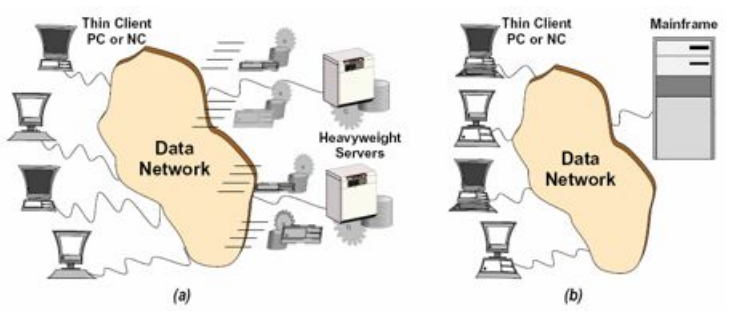
Diskless and X-Terminal Architect.
(on disk and/or CPU servers)



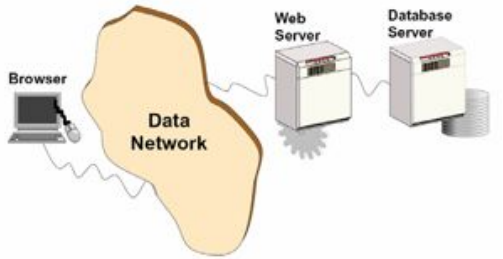
Client-Server Architectures
(Clients became "fat")



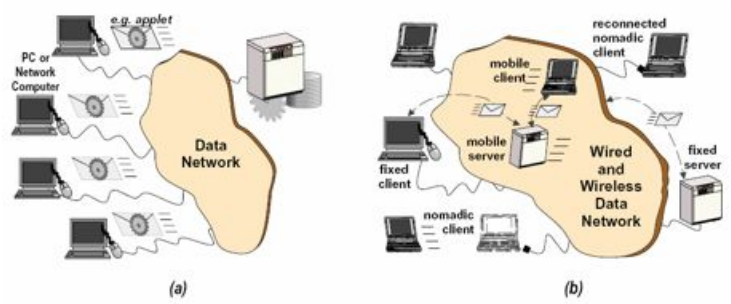
3-tier Client-Server or Thin-Client Archit.
(brought mainframes back, perhaps unnecessarily)



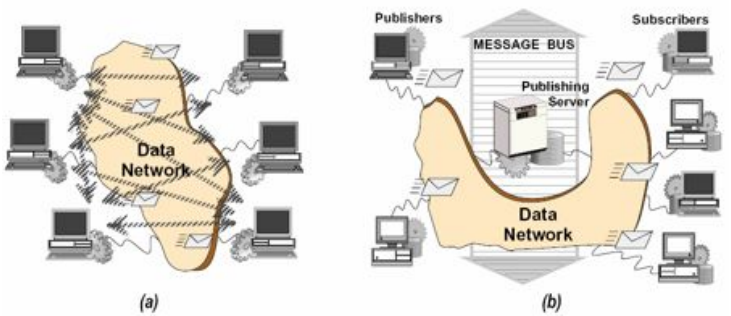
3-tier Client-Server Web-based Archit.



Mobile Code Architectures:
(a) Portable and Mobile Code; (b) Mobile Nodes



Event-based Architectures:
(a) Multipeer; (b) Publisher-subscriber



Non-functional properties of Distributed Systems

- What systems are, rather than what they do
- Some familiar names:
  - reliability and availability (fault tolerance)
  - timeliness and predictability (real-time)
  - confidentiality and integrity (security)

Some formal notions

Formal system predicates
a colloquial definition

- Objective
  - specify in a formal manner, including formulas containing logic (and, or, exists, forall), temporal logic (eventually, always) and time (until/from) operators
  - we can specify the properties of any program or protocol in terms of properties of: *safety and/or timeliness , liveness*

Formal system predicates
a colloquial definition

- Safety
  - the measure in which a service/program does not do bad things
  - safety properties specify that wrong events never take place
  - a safety property specifies that a predicate P is always true
  - example, "any delivered message is delivered to all correct participants" is a safety property
  - If it is not secured, the system becomes incorrect
  - however, it does not impose that messages are delivered at all

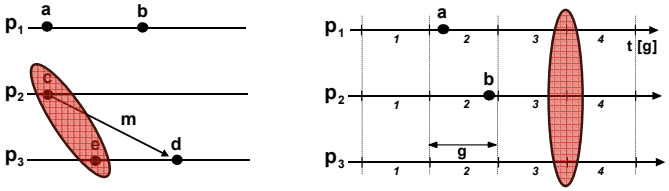
Formal system predicates
a colloquial definition

- Liveness
  - the measure in which a service/program does good things
  - liveness properties specify that good events eventually take place
  - a liveness property specifies that predicate P will eventually be true
  - example, "any message sent is delivered to at least one participant" is a liveness property
  - If it is not secured, the system may not progress (messages are not delivered)

Formal system predicates
a colloquial definition

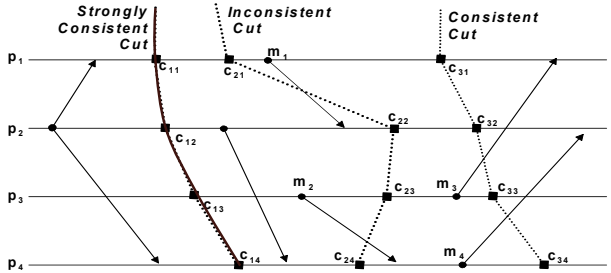
- Timeliness
  - a sub-class of safety property is timeliness, which specifies that a predicate P will be true in relation to a given instant of real time (until, before, at)
  - "any transaction completes until Tt from the start" is a timeliness property
  - to be secured, all transactions must execute within Tt time units

Space-Time and Lattice Diagrams



- Event types:
  - execution, send, receive, deliver
- Precedence:
  - a < b ; c < d
  - is c < e ?
- Time lattices:
  - based on notion of global time with granularity (tick) *g*
  - does 4:00 tick at exactly the same place everywhere?

Cuts and Global States (GS)



- Types of cuts:
  - inconsistent cut: snapshot gives invalid picture of GS
  - consistent cut: snapshot gives correct but possibly incomplete picture of GS (e.g., ignores messages in transit)
  - strongly consistent cut: snapshot faithfully represents GS

Distributed systems paradigms

Naming and addressing

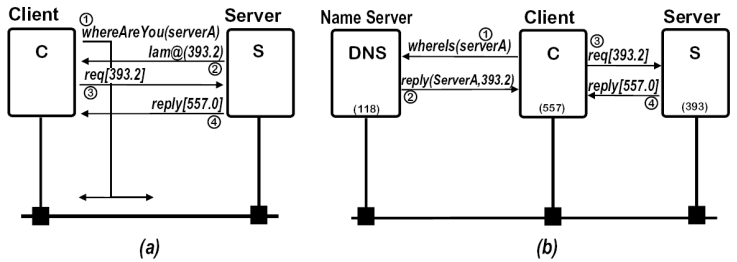


Figure 2.2. Name to Address Translation: (a) Broadcast; (b) Name Server

Message passing

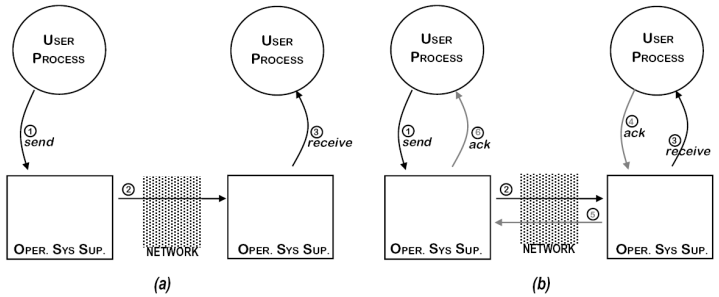


Figure 2.4. Message Passing Protocols: (a) Send-Receive;(b) Acknowledged-Send

Remote operations (contd.)

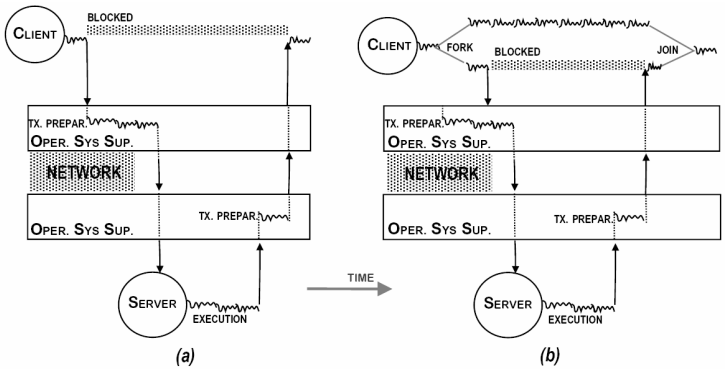


Figure 2.8. Remote Operation Interfaces: (a) Blocking; (b) Non-Blocking

Group communication

- process groups
- group communication service (multicast)
- group membership (views)
- main components of a multicast protocol:
  - routing;
  - omission tolerance;
  - flow-control;
  - ordering;
  - failure recovery.

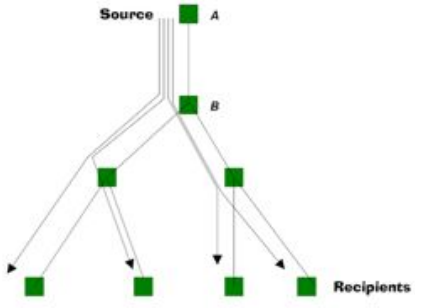


Figure 2.9. Multicast Tree

Time and synchrony

Time and Clocks

- Common uses of clocks in distributed systems:
- trigger events
- register the time at which events occurred
- measure durations
- artifact: support protocol implementation

Global Time

Why?

- trigger events
  - how do you synchronise distributed event triggering?
- register the time at which events occurred
  - how do you correlate distributed registers?
- measure durations
  - how do you measure what starts here and ends there?
- *Global Clock*:
  - abstraction: a set of mutually synchronised clocks

Absolute Time

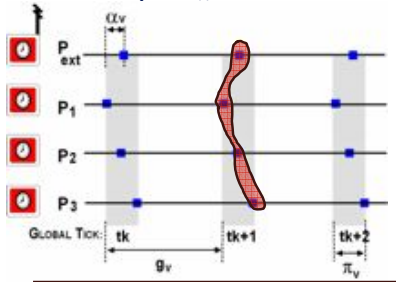
Why?

- coordination of systems that do not communicate directly
- bounding the error in lenghty duration measurement
- *Absolute global clock*:
  - abstraction: a set of clocks synchronised individually to a common reference
  - e.g., UTC- Universal Time Coordinated; TAI- Temps Atomique Internacional

Time and clocks

Properties of a Global Clock System

- Physical Granularity
  - $pc^{tk+1} - pc^{tk} = g$
- Virtual Granularity
  - $vc^{tk+1} - vc^{tk} = g_v$
- Convergence
  - $| vc_k(t^0) - vc_l(t^0) | \leq \delta_v$
- Precision
  - $| vc_k(t) - vc_l(t) | \leq \pi_v$, forall $0 \leq t$
- Rate
  - $1 - \rho_v \leq [\, t(vc_k^{tk+1}) - t(vc_k^{tk}) \,] / g_v \leq 1 + \rho_v$, forall $0 \leq tk < tk+1$
- Accuracy
  - $| vc_k(t) - t | \leq \alpha_v$, forall $0 \leq t$, t from an abs ref



So, does 4:00 tick at exactly the same place everywhere?

Time and clocks
Properties of a Global Clock System

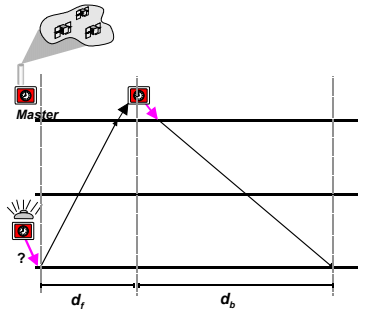
- Physical Granularity (g)
  - fundamental tick or pulse of hardware clock.
- Virtual Granularity (gv)
  - tick of the virtual clock, submultiple of g
- Convergence (δv)
  - measures how close virtual clocks are to each other immediately after the synchronization algorithm terminates.
- Precision (Πv)
  - measures how closely virtual clocks remain synchronized to each other at any time.
- Rate (ρv)
  - instantaneous rate of drift of virtual clocks.
- Envelope Rate ($\rho\alpha$)
  - long-term, or average rate of drift.
- Accuracy (αv)
  - measures how closely virtual clocks are synchronized to an absolute real time reference, provided externally.

Clock synchronisation

- Hardware clocks drift with time
  - some (e.g. cesium or rubidium GPS clocks) are extremely stable
  - but PC and workstation HW clocks are bad (worst than 1ppm)
- so they have to be synchronised periodically
  - *clock synchronisation protocols*
- Internal synchronisation:
  - ensures precision
  - normally clocks cooperatively readjust (*agreement or convergence based*)
- External synchronisation:
  - ensures accuracy and precision ($\pi_v = 2\ \alpha_v$)
  - normally clocks read from an external master (*round-trip-based*)

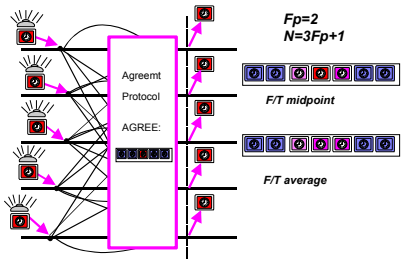
Clock synchronisation
(master- ou round-trip based)

- external synchronization :
  - based on round-trip measurement from central master clock
- Accuracy:
  - assessed by measuring round-trip delay
  - depends on delay symmetry
  - (df+db)/2 best run
- Precision:
  - precision is twice the accuracy ($\pi_v = 2\ \alpha_v$)



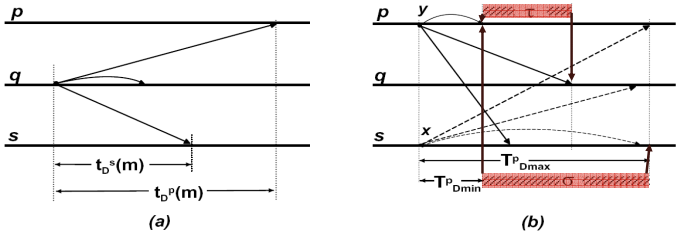
Clock synchronisation
agreement-based

- Agreement based:
  - convergence based on F/T average or median
  - clocks are Byzantine
- Precision:
  - precision depends on clock reading error
  - processors compute common value to set clocks to
  - time for agreement is in critical path of precision



Synchronism

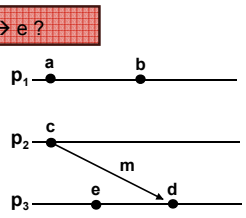
- synchronism means (w.r.t. messages):
  - known and bounded message delivery delay
- synchronism metrics of quality:
- Steadiness (σ)
  - $\sigma = \max_p(T_{Dmax} - T_{Dmin})$ (variance of delivery delay across execs)
- Tightness (τ)
  - $\tau = \max_{m,p,q}(t_D^p(m) - t_D^q(m))$ (variance of delivery delay in same exec)

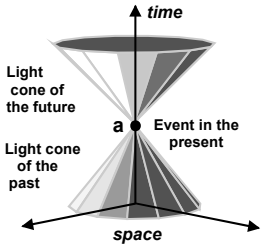


Ordering

Causal Order

So, is c → e ?





- **happened-before relation :**
  - ***a*** **--> *b* iff:**
  - ***a*** **before *b* locally**
  - ***a*** **send and *b*** **reception of *a***

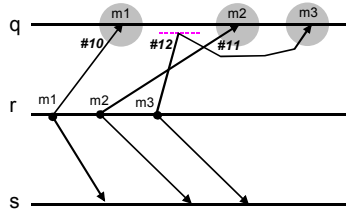
- cause-effect order:
  - natural universe order
- a partial order:
  - depends of time-like and space-like separation of events
  - relativistic effect due to speed difference between local and message events

Ordering

- Causal Delivery
  - for any two messages M1 and M2, sent by p and q, delivered to any correct processes, if *send(M1) --> send(M2),* then *deliver(M1) --> deliver(M2)*
  - Example: clients compete over a server to schedule a trip, buy some stock, and communicate between them at the same time; only causal reflects the inter-client relations on the server requests
- FIFO Delivery (*first-in-first-out*)
  - for any two messages M1 and M2, sent by p, delivered to any correct processes, if *send(M1) --> send(M2),* then *deliver(M1) --> deliver(M2)*
  - Example: this is a reduction of the general causal order to messages originated from only one sender (e.g. TCP ordering)

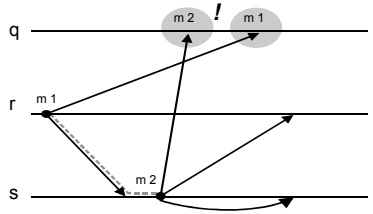
operations in FIFO order

(the most intuitive ordering)



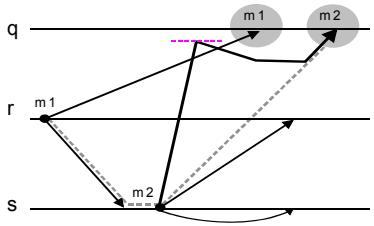
- r is solving problem by executing 3 modules in sequence
- he disseminates intermediate results (m1, m2, m3) to s and q, who perform the second phase, which depends on the sequence order.
- q got m1 with #10 and then m3 with #12, he knows m2 with #11 is missing and **delays** delivery of m3 until m2 arrives and only then it delivers messages m2 and m3 in that sequence.
- NB: in complex protocols, **reception** often different from **delivery**

when FIFO order is insufficient



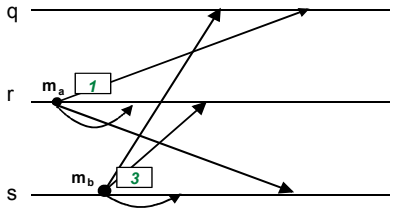
- problem was complex, so r breaks his job in steps, asking s to perform step 2 after he does step 1, which he signals with m1
- s executes step 2 when m1 arrives, after which it send m2
- problem: m1 got delayed, it will be delivered to q after m2
- since q waits for messages in the order they were issued to perform the second phase, the **application fails**
- what went wrong is that FIFO protocol does not capture m1 -> m2 causal relation and order inversion takes place
- cannot be used if competing senders also exchange messages

Solution: causal order



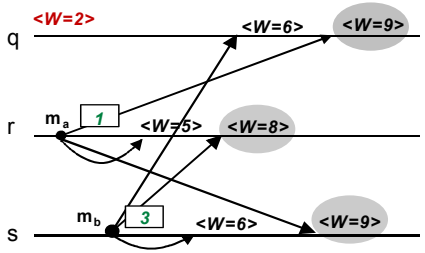
- FIFO is expanded to causal and englobes all nodes: m1 -> m2 is now recognised
- m1 is delayed to q, but q delays delivery of m2, to fulfil causal delivery

operations in causal order



- r leads a team work performing some computations
- result is accumulated in variable W, update function compares W previous state to new result, takes greatest and adds 3
- errors in previous works make r request all steps done in parallel by r, q and s, and results disseminated to all, to compare results and replicate W. Any one finishing a step posts result to all including himself, in **causal order**.
- if everybody is doing the same steps, it is expected for W to be the same everywhere.

when causal order is insufficient

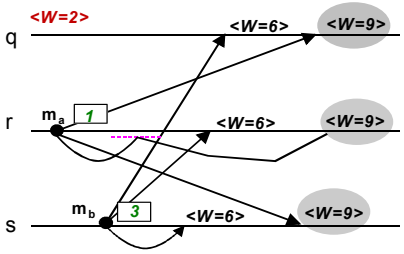


- initially W=2, and r and s disseminate their results concurrently
- so causal order protocol does not order them:
- ma = <1> is received first at r, W=2+3=5, then mb = <3> is received and W=5+3=8. mb = <3> is received first at q, W=3+3=6, then mb = <3> is received and W=6+3=9
- this violates replicated computation correctness subsequent steps depending on the value of W will not be consistent

Total ordering implementations

- Total ordering
  - Any two messages delivered to any pair of participants are delivered in the same order to both participants
  - Example: sending operation or update requests to replicas of a server, so that they execute them inthe same order and produce the same result and/or assume the same state

Solution: total order

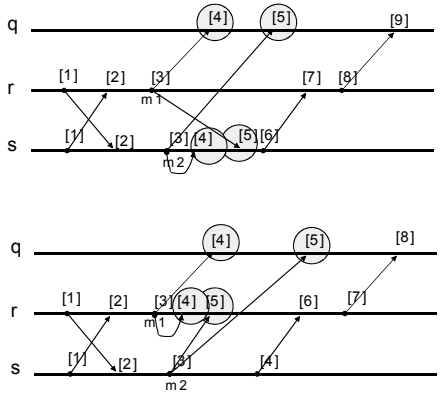


- previous problem is solved with total order
- active replica management requires total order, be it causal or not
- in which cases can we do active replicas without total order ?
- *think....*

Ordering mechanisms and algorithms

Causal ordering with logical clocks

- ***Objective:***
  - **order events by cause-effect or precedence (e1 --> e2)**
- ***Implementation rules:***
  - **initially: LCi=0 forall i;**
  - **at each *e* local to *p*: incr. LCp**
  - **at each send, timestamp *m* with LC value: LC(m) = LCp**
  - **at each reception at *q*, incr. LCq, and update LCq with max [LCq,LC(m)]**
- ***Ordering rules:***
  - **e1 --> e2 ==> LC(e1) < LC(e2)**

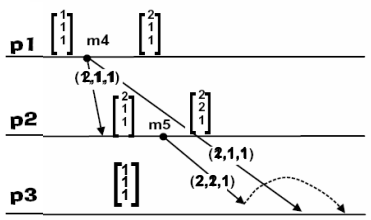


Causal ordering with vector clocks in action

(ISIS CBCAST protocol)

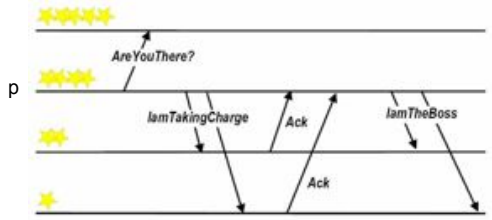
- ***Implementation rules (Pi -> Pj):***
- **initially:** VTi[k]=0 forall i,k
- ***at each send m*:**
- incr. VTi[i] and timestamp *m* with VTi: VT(m)
- ***at each Rx at Pj delay delivery until*:**
- msg m-1 from Pi seen : VT(Pj)[i]=VT(m)[i]-1 and
- all msgs preceding m delivered at Pi,

  also delivered at Pj : VT(m)[k]≤VT(Pj)[k] , forall k
- ***Pi does not delay msgs to itself***
- ***upon each delivery:***
- update VT(Pj) w/ max[VT(Pj)[k],VT(m)[k]] forall k
- ***Example:***
- **when p3 receives m5, VT3=[1,1,1]**
- **since VT(m) =[2,2,1], msg from p1 is missing at p3**
- **wait until rx m4, which sets VT3=[2,1,1], and deliver m5, setting VT3=[2,2,1]**



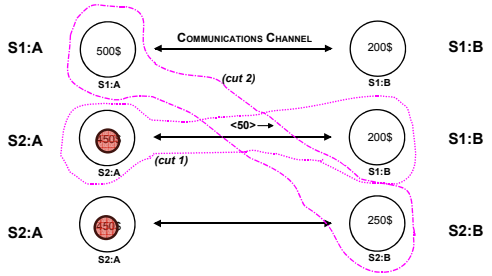
Coordination and Consistency

Leader election *(bully algorithm)*



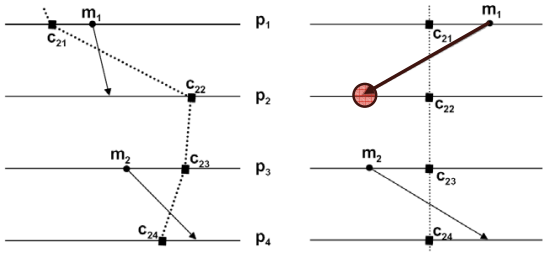
- always elects the highest ranked active candidate (i.e., process with the lowest identier)
- instead of sending a message to every process, process p trying to become leader just sends *AreYouThere?* to higher ranks, if someone replies, p silently gives up
- if nobody replies, p tells all lower ranks of his intention, sending *IAmTakingCharge*, and waits for ack from each
- when all acks come (or a timeout occurrs, since some of the processes with lower rank may have crashed), p assumes the leadership by sending *IAmTheBoss* to all processes.

Consistent Global States
(issues with Ad-hoc State Snapshots)



- total=700$; money transfer A-> B, 50$; during transfer, ad-hoc snapshot is done, from external node sending messages to A and B.
- cut 1: <S2:A=450$, S1:B=200$>=650$! (msg sent, not received!)
- cut 2: <S1:A=500$, S2:B=250$>=750$! (msg received, not sent!)
- a correct snapshot protocol will flush the channels to ensure consistency

Inconsistent Cuts



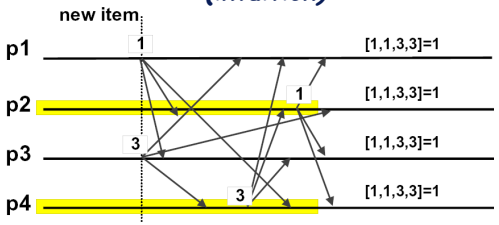
- what you get if you don't use the right algorithm:
  - is there anything strange with message m1?
  - and now?
  - a correct snapshot protocol will discard messages "not sent" to ensure consistency

Consensus

- Validity
  - if a process decides v, then v was proposed by some process
  - no process decides more than once
- Agreement
  - No two correct processes decide differently
- Termination
  - Every correct process eventually decides

- Consensus is equivalent to atomic broadcast
  - That is, one can implement one with the other
  - Does not mean that all such implementations are efficient!

Distributed consensus
(intuition)



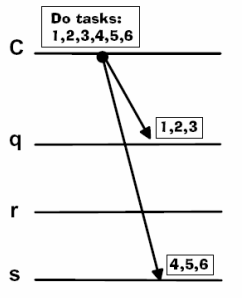
- set of processes must agree on one action, in a decentralized way: decide who keeps each new item that arrives, all send their votes to all
- new item arrives, p2 and p4 are busy, so only p1 and p3 offer to pick it
- p2 receives the proposal from p1 in the first place, thus it supports p1, while for the same reason p4 supports p3.
- when all votes are collected, all have same vector to decide from (1,1,3,3)
- any agreed deterministic function will do, ex:
  - "winner is the one with more votes, in case of tie, the smaller ID wins"
- the item is assigned to p1.

Membership

- group membership
  - set of processes belonging to the group at a given point in time
- membership service:
  - keeps track of membership and provides info to group members
- group view:
  - subset of members mutually reachable at a given point
- group membership is often dynamic:
  - in response to user demand or changes in the runtime environment (load, failures, etc)
  - it may grow, by letting new processes join the group
  - it may shrink, by letting members leave the group
  - view changes when processes fail or when they recover

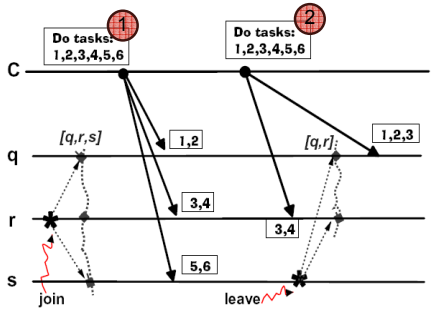
Agreement on Membership
(decentralized applications)

- coherent notion of membership useful for a number of applications
- e.g. decentralized dispatcher
  - group of workers (set of parallel processors) divide a task requested by client by the current number of elements
  - dispatch is local, split is dynamic: processors may come and go



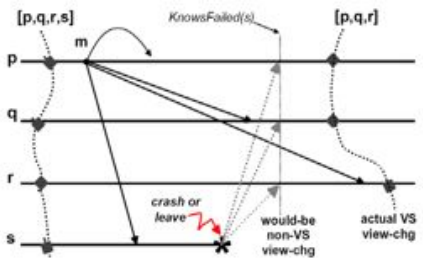
Agreement on Membership
(issues with ad-hoc view change)

- r joins group, view is {q,r,s}, then s leaves
- change notification not consistent:
  - r gets request 2 in view {q,r,s}, so picks <3,4>
  - q gets request 2 in view {q,r}, so picks <1,2,3>
- what went wrong:
  - <3> is performed twice
  - <5,6> are not performed



View Synchrony
view-synchronous view change

- solution to previous problem:
  - membership changes notified consistently with message flow !
  - if a message m is delivered to a process p in view Vi, then for all q in Vi, m is also delivered to q in view Vi.
- how to ensure all processes deliver same messages in same view?
  - flush messages until a consistent cut

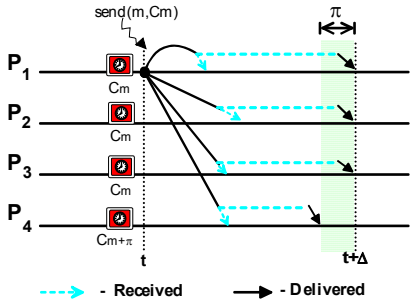


Atomic Broadcast
properties

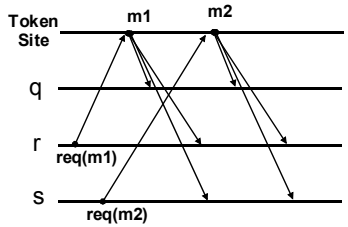
- Validity
  - If a correct processor broadcasts a message M, then some correct processor eventually delivers M.
- Agreement
  - If a correct processor delivers a message M, then all correct processors eventually deliver M.
- Integrity
  - For any message M, every correct process *p* delivers M at most once
  - If process *p* delivers M and *sender*(*M*) is correct, then M was previously broadcast by *sender*(*M*).
- Total order
  - If two correct processors deliver two messages *M*1 and *M*2 then both processors deliver the two messages in the same order.

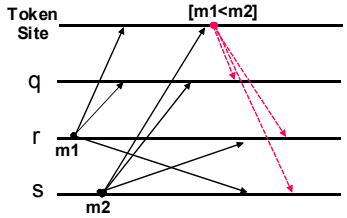
Atomic Broadcast
(symmetric approach - intuition)

- reliability through space redundancy or tx-w-resp
- total causal ordering through physical clock timestamps
- delivers by message timestamp order
- disambiguates e.g. by UID or MAC address, etc.
- i.e. msg(Cm) after Cm-1 and before Cm+1 everywhere
- can be done with logical timestamps



Atomic Broadcast
(asymmetric approach - intuition)





- reliability by tx-w-resp w/ store-and-forward or diffusion w/ negative ack
- token-based: sequencer decides ordering and propagates to all
- total non-causal order

Replicated computations

- distributed applications may run replicated pieces of code which should behave in the same way (e.g. fault tolerance, performance)
- atomic broadcast guarantees, in a decentralized way, that replicas receive the same sequence of inputs:
  - same requests, in the same order

Concurrency and Atomicity

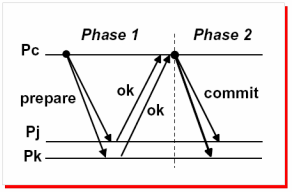
Atomicity

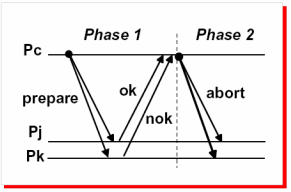
- atomicity is the property of an indivisible operation
- transactional atomicity is that property extended to a set of operations that are made look like indivisible
- atomic transaction is an operation exhibiting that property
- several techniques concur to achieve it:
  - either all operations are performed, or the whole transaction is aborted
  - intermediate results cannot be seen before the end
  - results must be stored in non-volatile memory to be persistent

Distributed Atomicity

- how to ensure atomicity of transactions that run across several nodes?
- Problem:
  - partial failure of nodes, and partitions, leading to inconsistent termination
- Solution:
  - distributed atomic commitment
  - most used protocol - *two-phase commit*

2-phase Distributed Atomic Commitment





- Two-phase commit
  - commit
  - abort
- Problem:
  - subject to blocking

Distributed systems models

Synchrony models

Table 3.2. Synchronous Model Properties

• Processing delays have a known bound
• Message delivery delays have a known bound
• Rate of drift of local clocks has a known bound
• Difference between local clocks has a known bound

- The last property species the existence of synchronized clocks.
- Whilst not required of every synchronous system, the first three properties make it possible.

Synchrony models

Table 3.1. Asynchronous Model Properties

• Processing delays are unbounded or unknown
• Message delivery delays are unbounded or unknown
• Rate of drift of local clocks is unbounded or unknown
• Difference between local clocks is unbounded or unknown

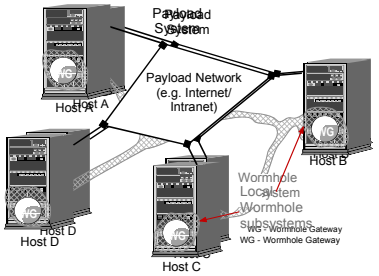
- The last two are essentially equivalent:
- since a local clock in a time-free system is nothing more than a sequence counter, synchronized clocks are also impossible in an asynchronous system.
- however, they are listed for a better comparison with synchronous models.

Synchrony models in between

- *synchrony or asynchrony are not homogeneous properties of systems*:
  - they may vary with time
  - they may vary with the part of the system being considered
- that is, sometimes
  - the system is not always asynchronous, and/or
  - the system is not asynchronous everywhere
- intermediate models attempting best of both worlds
  - asynchronous with failure detectors
  - timed asynchronous
  - quasi-synchronous
  - wormholes

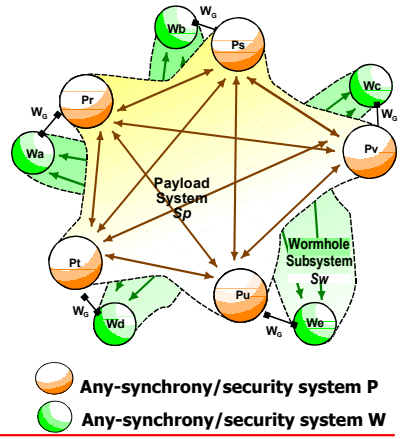
Wormhole models

- New design philosophy for distributed systems:
  - constructs with privileged properties which endow systems with the capability of evading the uncertainty of environment ("taking a shortcut") for certain crucial steps of their operation,
  - this allows achieving some predictability (the required "hard properties")
- Based on hybrid distributed systems models

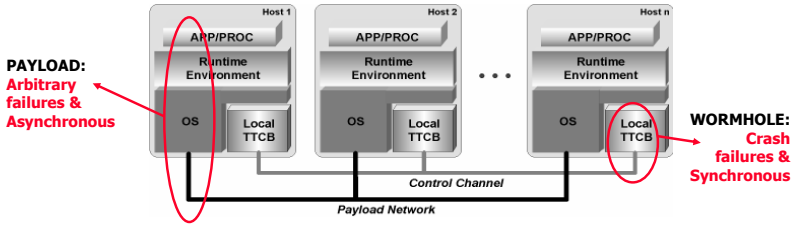


Hybrid distributed systems models
(a.k.a. Wormhole models)

- models in between
  - system is neither synchronous nor asynchronous
  - it is both, at the same time, different places
- concept may be extended to fault model:
  - system is neither fail-silent (crash) nor fail-arbitrary (Byzantine)
  - it is both, at the same time, different places



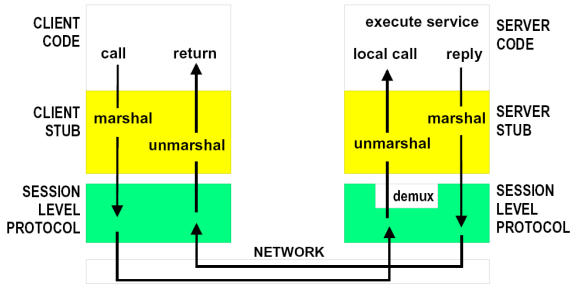
Example system with wormhole



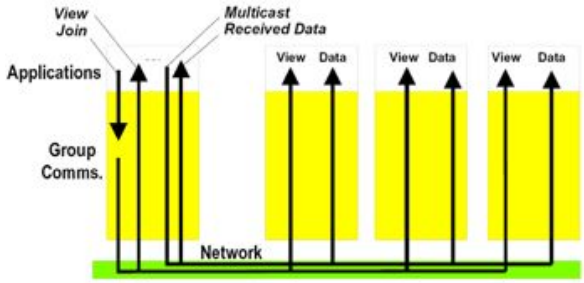
- the example system (TTCB) is a distributed real-time and security kernel providing a minimal set of trusted/timely services
  - failure detection
  - local authentication
  - agreement on a fixed sized block of data (TBA)
  - trustworthy global timestamps and random numbers

Distributed Computing models

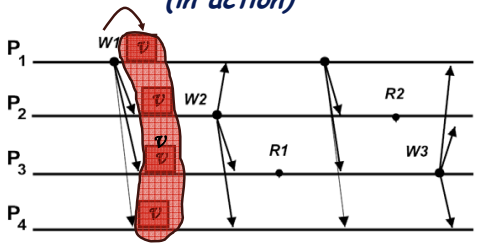
Client-server with RPC
(in action)



Group-oriented
(in action)

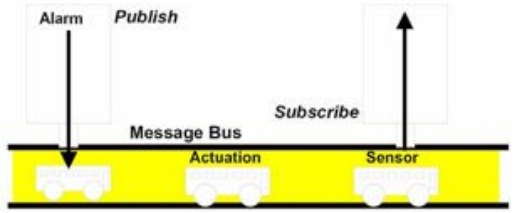


Distributed Shared Memory (DSM)
(in action)



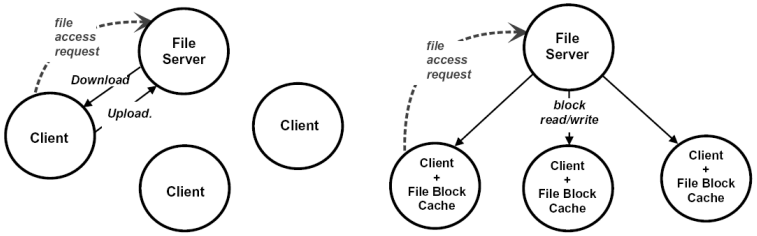
- simulating a shared data item in local memory, over a distributed and/or possibly replicated memory

Message bus
(in action)



- publisher-subscriber
- oriented to event processing
- supported by event-based communication

Distributed File systems
(in action)



- Download-Upload
  - retrieving and storing the whole file from/to the file server
- Remote Access
  - clients cache file blocks that they request from the server
  - files remain remote on the server

Object-oriented
(in action)