Seminar on Reliability in Distributed Systems

Reliable and highly-available Systems

The following slides are mostly based on „Reliable distributed systems“ by Kenneth P. Birman. The quotes are also taken from there.

Some Quotes on Reliability

„Unfortunately, for three decades, the computing industry has tried (and failed) to make the mechanisms of reliable, secure distributed computing transparent“ (xxi)

„There just isn‘t any way to hide the structure of a massive and massively complex distributed system scattered over multiple data systems. We need to learn to expose system structure, to manage it intelligently but explicitly and to embed intelligence right into the application so that the application can sense problems.“

„When reliability also entails high availability we are in trouble“

„Neither .NET nor J2EE really has the features needed to promote high-availability or other kinds of quality of service guarantees.“(234)

The goal is to increase reliability and availability even with systems NOT initially built this way – because they are the reality today

RAS

Reliability

Availability

Security

Again, correctness is not just consistency. It includes LIVENESS. A mission critical system needs to make progress as well.

Reliability Engineering – what does it take?

-Security engineering courses

-System management courses

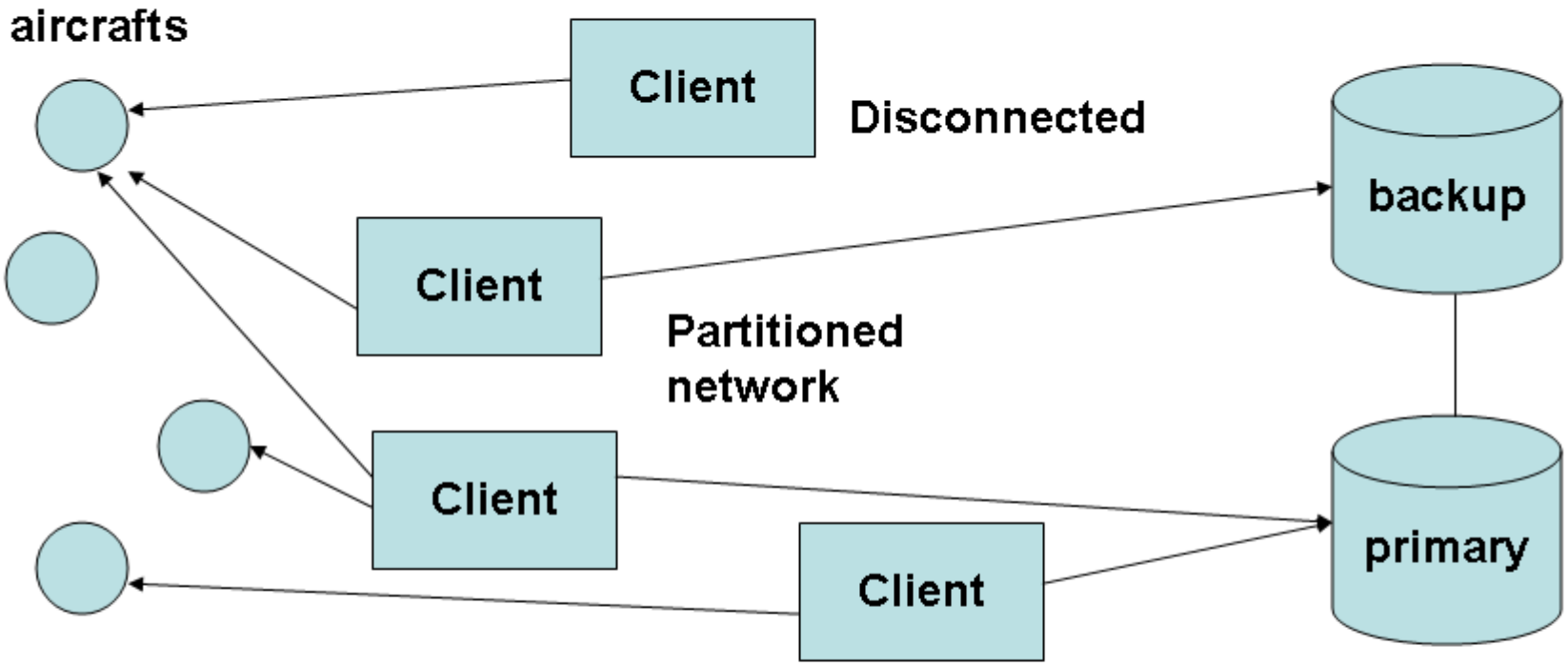
-Large scale architecture and structure courses

-Modeling courses

-Risk analysis and management courses

But most important according to Birman is the development of a „reliability mindset“ that simply keeps the non-functional requirements always close to ones mind. It starts simply with asking „what happens to availability if component XYZ does not respond/run/work anymore?“ And „how do I detect such a failure?“

Partitioning in a mission critical application



Strategy: just wait till all components are up and running again and retry requests until then. All components are assumed to restart after some time successfully.

But: this strategy does not guarantee LIVENESS of your real-time application. See transaction protocols later. (Birman XXV)

Reliability Questions

-What is the role of replication in reliable systems?

-What is needed to achieve safe replication?

-How are failures detected and why is this important?

-How are systems updated while still being accessible?

-What happens if failures occur during updates?

-How do new processes join a group and what is the role of group membership?

Failure Places

-Network: partitioning

-CPU/Hardware: instruction failures, RAM failures

-Operating System: Crash, reduced function

-Application: crash, stopped, partially functioning

Unfortunately in most cases there is no failure detection service which would allow others to take appropriate action. Such a service is possible and could detect even partitionings etc. through the use of MIBs, triangulation etc. Applications could track themselves and restart if needed.

Failure Types

-Bohr-Bug: shows up consistently and can be reproduced. Easy to recognize and fix.

-Heisenbug: shows up intermittently, depending on the order of execution. High degree of non-determinism and context dependency

Due to our complex IT environments the Heisenbugs are both more frequent and much harder to solve. They are only symptoms of a deeper problem. Changes to software may make them disappear – for a while. More changes might caues them to show up again. Example: deadlock „solving“ through delays instead of ressource order management.

Failure Models

-Failstop: A machine fails completely AND the failure is reported to other machines reliably.

-Byzantine Errors: machines or part of machines, networks, applications fail in unpredictable ways and recover partially.

Many protocols to achieve consistency and availability make certain assumptions about failure models. This is rather obvious with transaction protocols which may assume failstop behavior by its participants if the protocol should terminate.

Failures and Timeouts

A timeout is NOT a reliable way to detect failure. It can be caused by short interruptions on the network, overload conditions, routing changes etc.

A timeout CANNOT distinguish between the different places and kinds of failures.

It CANNOT be used in protocols which require failstop behavior of its participants

Most distributed systems offer only timeouts for applications to notice problems

A timeout does not allow conclusions about the state of participants. It is unable to answer questions about membership (and therefore responsibility). If timeouts ARE used as a failure notification „split-brain“ condidtions (e.g. airtraffic control) can result (Birman 248)

Replication and Membership

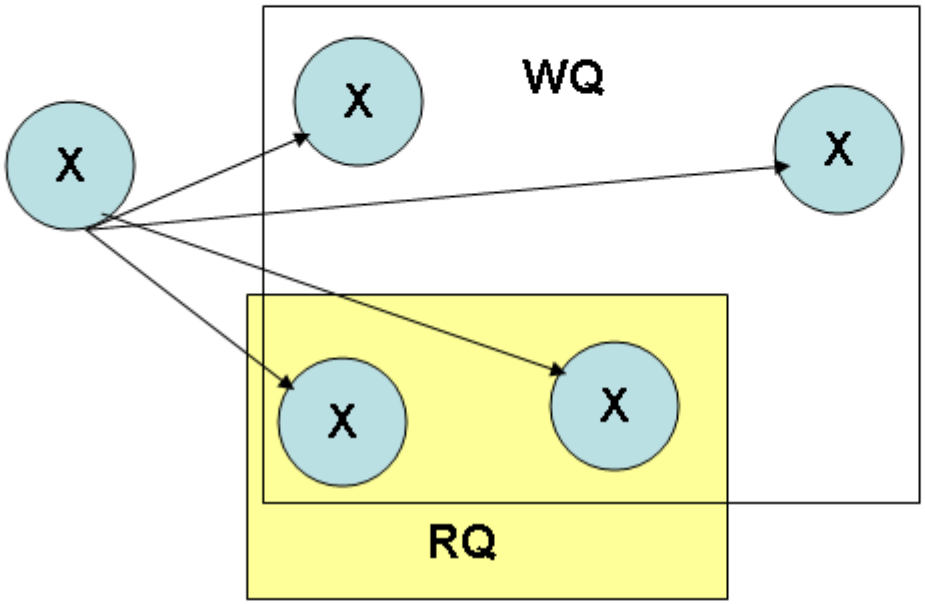
Replication of Data

Replication of Processing

Replication is at the core of reliability and availability. It absolutely requires a clear and performant way to decide about the membership within a replication group. Who is replicating what in which location?

The question of membership includes other problems in distributed systems: consistency, agreement, availability and performance

Static Membership: Quorum Update and Read Arc.



Majority rulez! Not all known processes N that replicate a value X need to be reached in every update. But the number of processes to read (Read Quorum RQ) and the number to write (Write Quorum WQ) need to be at least N+1. E.g. RQ==2 and WQ==4 in a system of N==5

Scalability of Quorum Update and Read Arc.

-The write quorum needs to be smaller than the number of processes to achieve fault tolerance. This implies that the number of reads JUST TO READ a variable needs to go up.

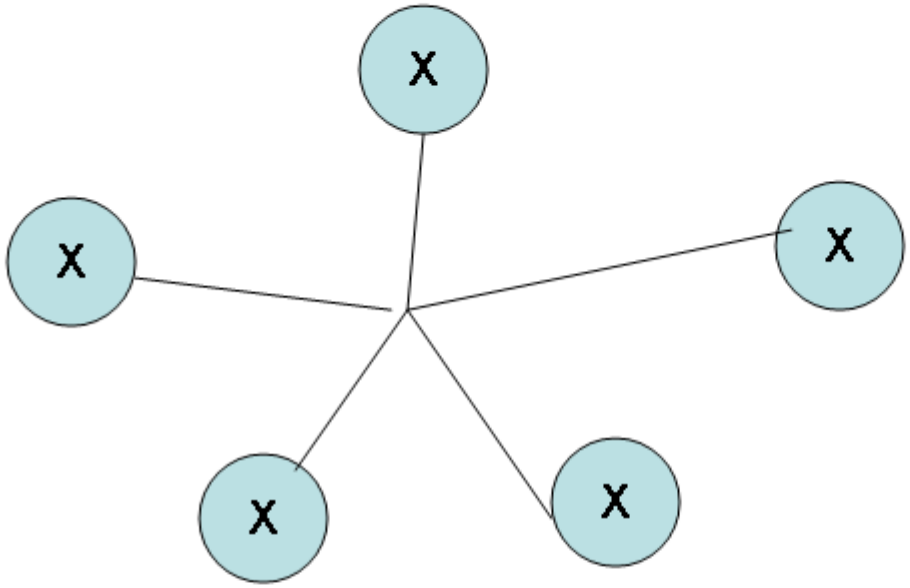
-Reading X is therefore tied to the speed of RPCs to other processes.

-Communication overhead through pending operations slow the system down by O(n-square)

-Updates require a vote/agreement from several machines. The current values is decided through the latest timestamp

Systems with fixed, known membership are quorum based. A majority of participants needs to see the operation before it can be committed. Agreement is reached through vote collection.

Dynamic Group Membership Services



Multicast based membership service

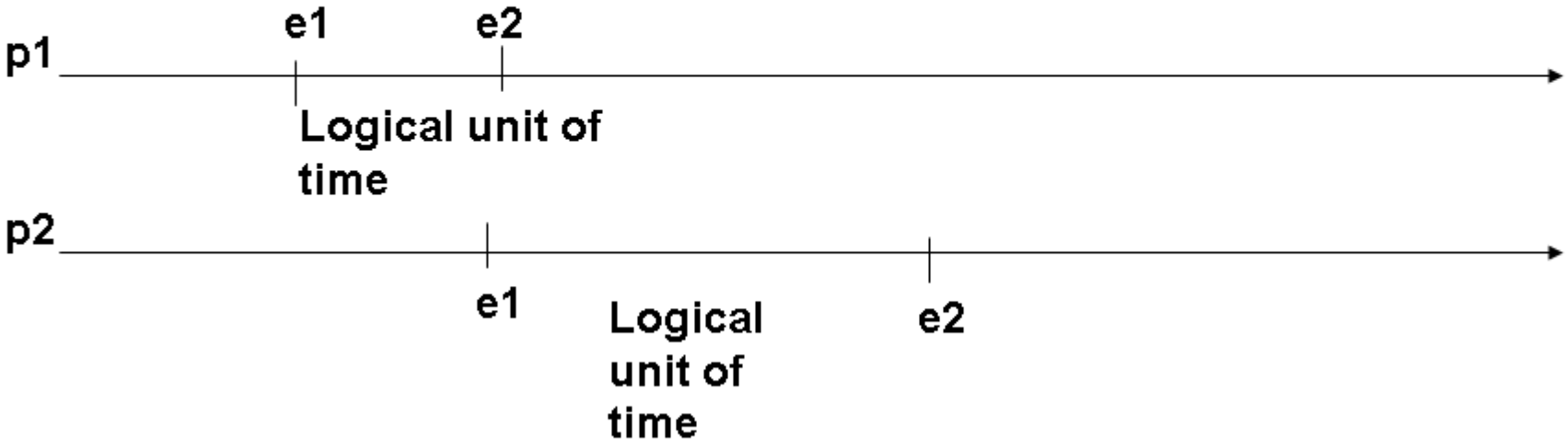
Examples are: Horus, Spread. (Birman 251ff). Scales almost linearly up to 32 members. 80000 updates in a group of five (no disk access, failstop nodes). Application does not wait for acknowledge.

Time in Distributed Systems

- No global time
- logical clocks
- vector time

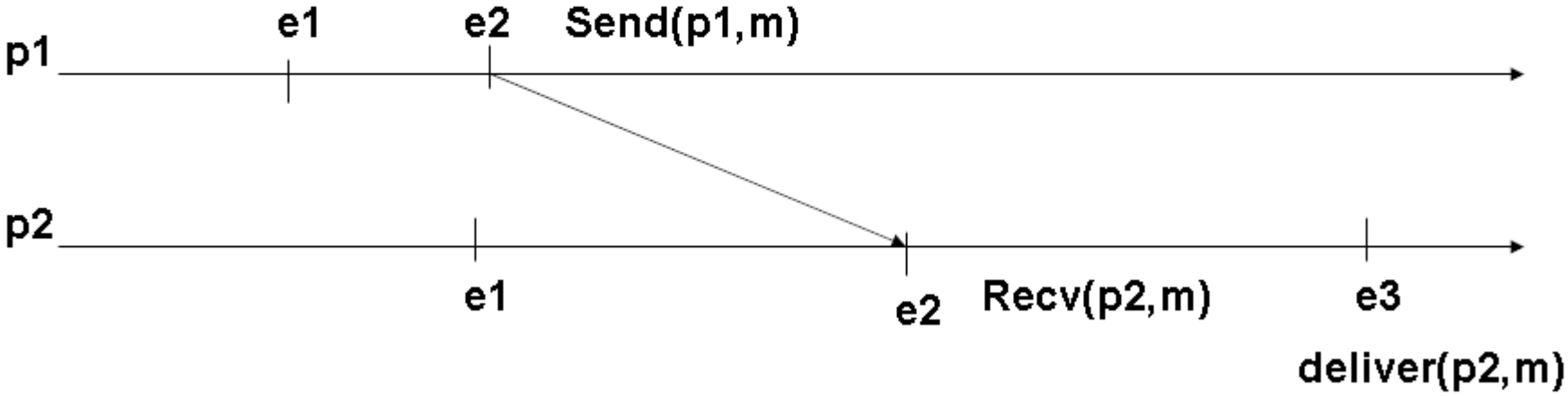
There is no global time in distributed systems. Logical time modelled as partially ordered events within a process or also between processes.

Processes, Time and Events: internal time



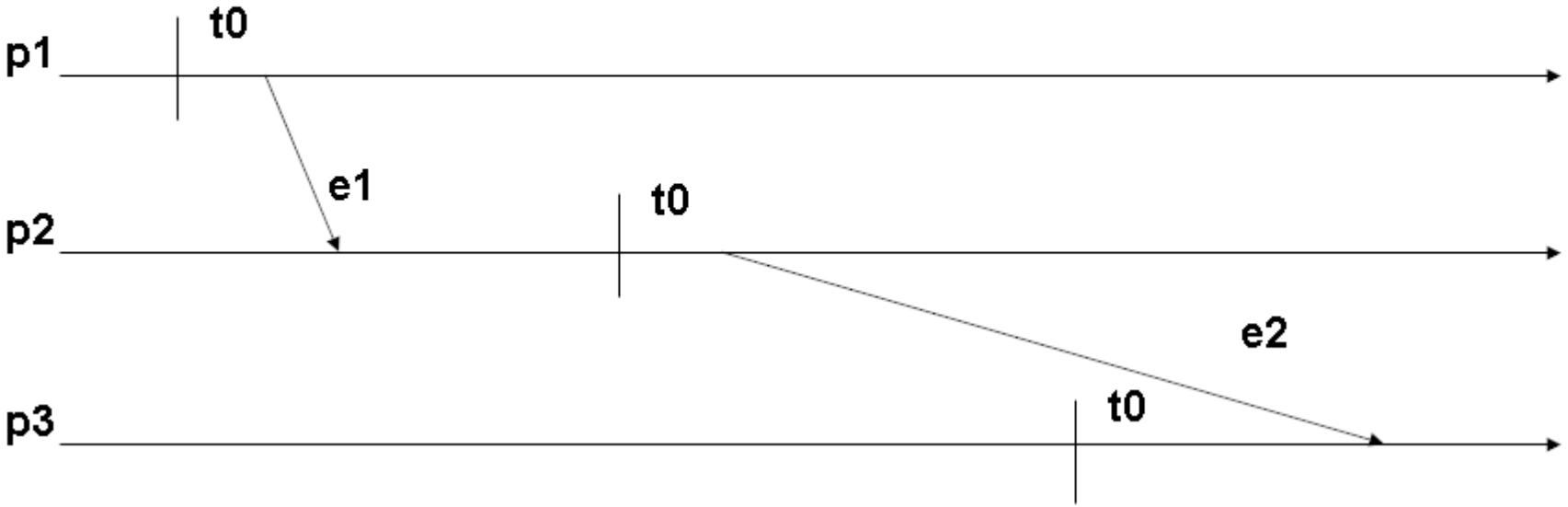
Events are partially ordered within processes according to a chosen causal model and granularity. E1 < e2 means e1 happend before e2. The time between events is a logical unit of time.

Processes, Time and Events: external events



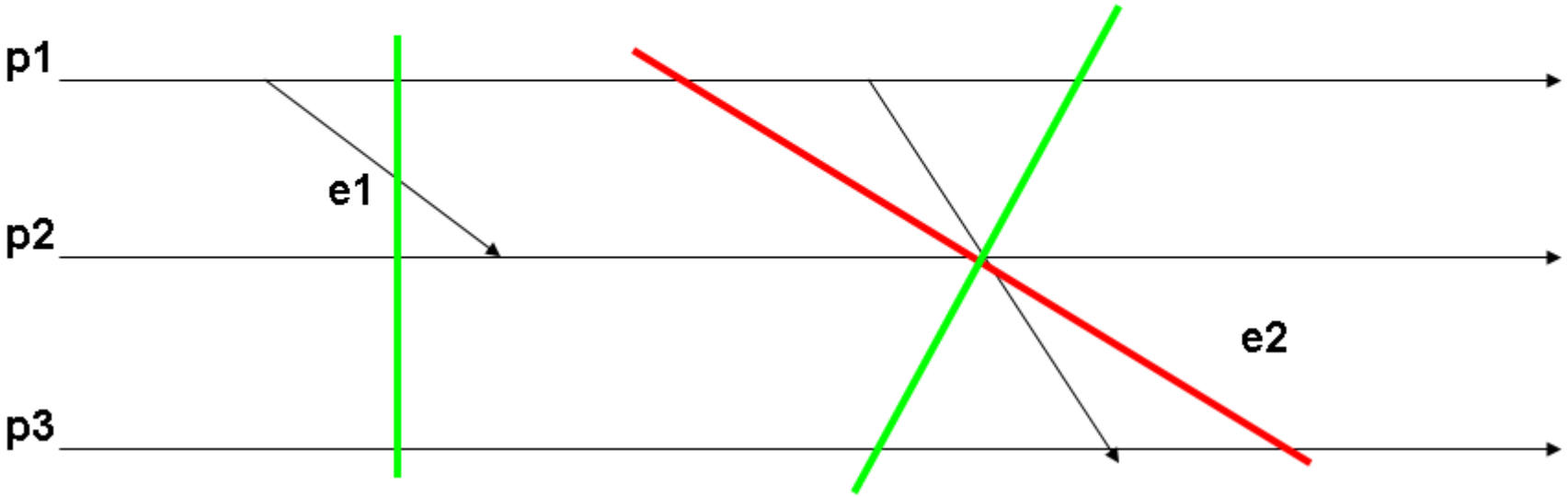
Events delivered through messages clearly relate processes and their times and events. This external order is also a partial order of events between processes: send(p1,m) < recv(p2,m)

No Global Time in Distributed Systems



The processes p1-p3 run on different clocks. The clock skew is visible in the distances of t0 on each time line. T0 represents a moment in absolute (theoretical) time in this distributed system. For p2 it looks like e1 is coming from the future (the sender timestamp is bigger than p2‘s current time). E2 looks ok for p3. Causal meta-data in the system can order the events properly. Alternatively logical clocks and vector clocks (see Birman) can be used to order events on each process and between processes. This does NOT require a central authority (like an event system for CEP can provide)

Consistent Cuts vs Inconsistent Cuts



Consistent cuts produce causal possible events. With inconsitent cuts (red) some events arrive before they have been sent. Consistency of cuts is independent of simultaneous capture. (Birman 257)

Distributed Commit

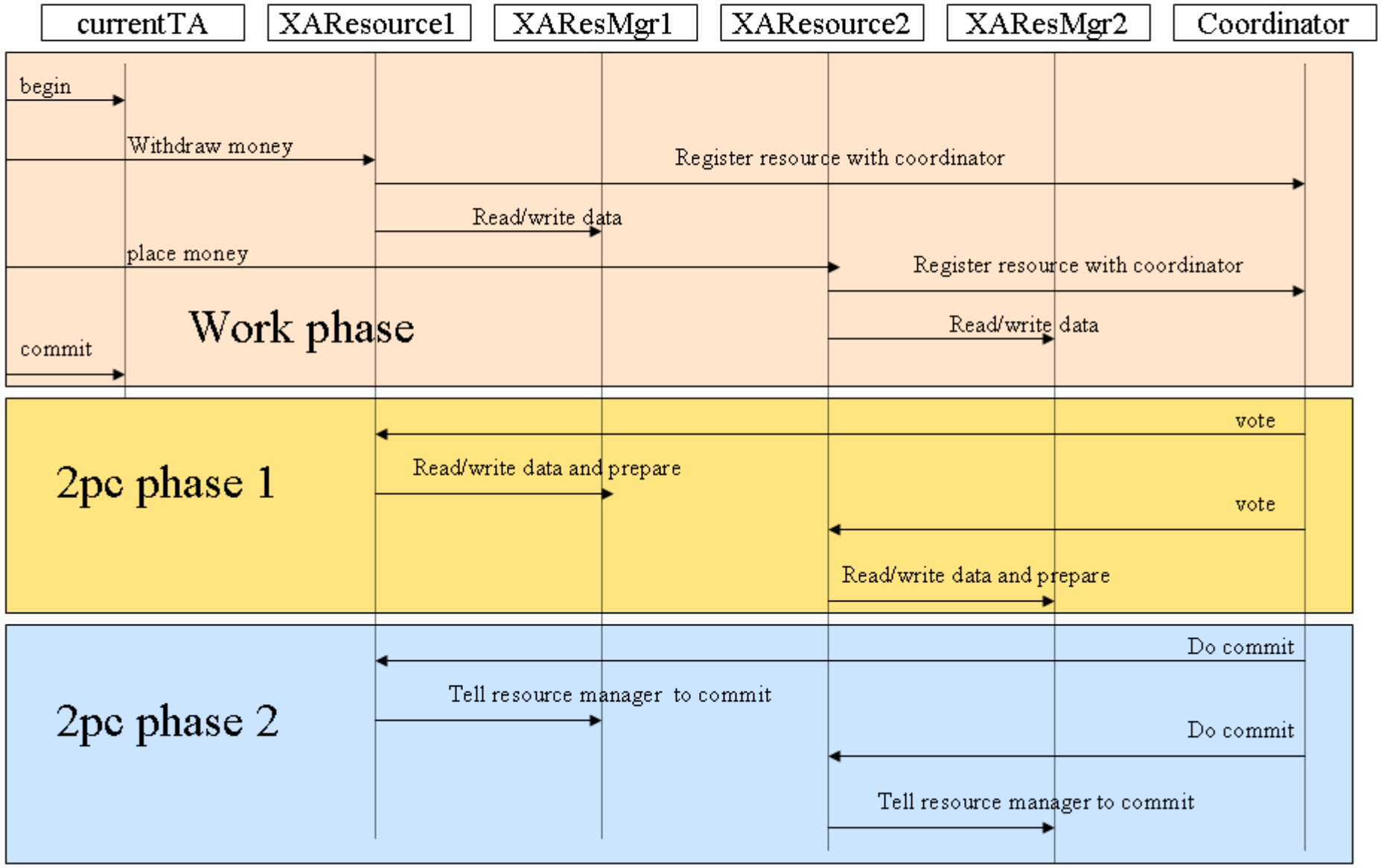
2-Phase Commit

2-Phase Commit with communicating participants and garbage collection

3-Phase Commit protocol

Goal is to understand the assumptions behind those protocols and how they are affected by failure models. Can they reach agreement or termination in any case? This will lead over to the „FLP Impossiblity result“

Example of 2PC



Failure models in 2PC

Work phase:

- A participant crashes or is unavailable in work phase.

The coordinator will call for a rollback.

- The client crashes in work phase (commit is not called). Coordinator will finally time-out the TA and call rollback.

Voting Phase:

- If a resource becomes unavailable or has other problems, the coordinator will call rollback

Commit Phase: (server uncertainty)

- a crashed server will consult the coordinator after re-start and ask for the decision (commit or rollback)

Assumptions: Participants will retry forever. Participants will restart eventually. If the coordinator crashes no progress can be made after voting.

Special problems of distributed TA's

- **Resources: Participants in distributed TA's use up many system resources due to logging all actions to temporary persistent storage. Also considerable parts of a system may get locked during a TA.**
- **Coordinator – a single point of failure? Even the coordinator must prepare for a crash and log all actions to temporary persistent storage.**
- **Heuristic outcomes for transactions. Under certain circumstances the outcome of a transaction may only follow a certain heuristic because the real outcome could not be determined. (see exercises)**

2PC with communicating participants

Requires third phase for garbage collection. Can still block if coordinator and one participant crash.

Resources

-Guerrai et.al. Introduction to reliable distributed programming

- Ken Birman, Reliable Distributed Systems