# Fault Tolerance and the Five-Second Rule

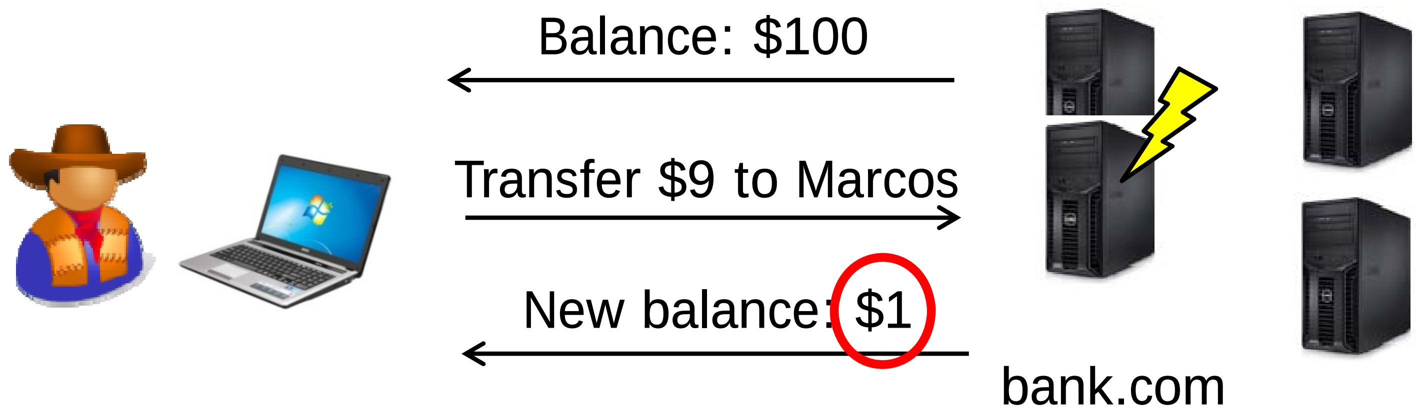
Ang Chen    Hanjun Xiao

**Andreas Haeberlen**    Linh Thi Xuan Phan

Department of Computer and Information Science

University of Pennsylvania

# Faults in Distributed Systems




- Nodes in a distributed system can fail
  - Example: Online banking
- The consequences can be serious
  - Example: Monetary loss
- Solution: Use fault-tolerance techniques

# Faults in Real Life


- Transactions in real life can fail, too!
  - Example: Paying with cash at the checkout counter
- Failures can have bad consequences
  - Example: Getting shortchanged
- Solution: Use fault-tolerance techniques?

# Online vs. offline

- How do we do handle this in the real world?
  - No masking: The transaction is allowed to fail initially
  - Detection: Participants check the results
  - Recovery: Detected failures are fixed if possible
  - Timeliness: Checking happens quickly (to limit damage)

- Can we do the same in distributed systems?

- Our proposal: Bounded-time recovery (BTR)
  - Intuition: When a node fails, the system may make mistakes <u>for a limited time</u> (e.g., 100ms), but then it recovers
  - Should be a provable property - not just best-effort!

# When would BTR be sufficient?

- Not all systems can use BTR
  - Example: Systems where failures are immediately fatal

- But there are systems that could benefit!
  - Example: Cyber-physical systems
  - Physical part often has some inertia
  - Control algorithms can often tolerate some mistakes
  - Time bound is key: Fixing problems 'eventually' is not enough!

# What could we gain from BTR?

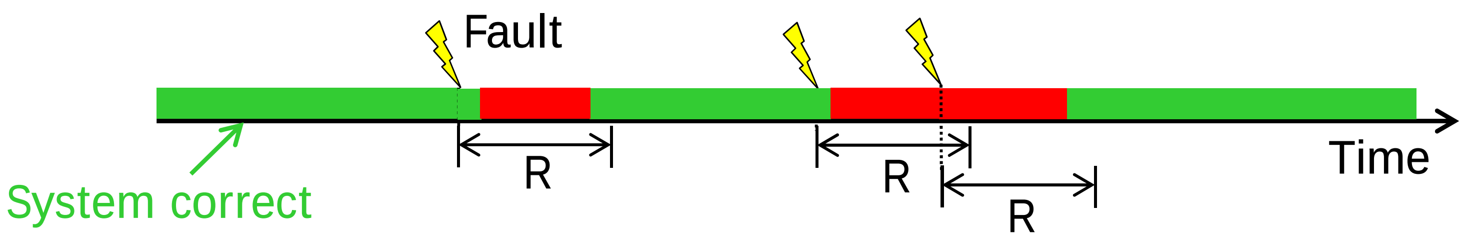
- Opportunity #1: Lower cost
  - Detection is cheaper than masking
  - Particularly important for CPS
- Opportunity #2: Timing guarantees
  - Even most BFT solutions cannot guarantee timely responses when the system is under attack
- Opportunity #3: Fine-grained responses
  - Typical fault-tolerance guarantee is "all or nothing"
  - BTR can recover failures in many ways, e.g., by dropping less important tasks or by adjusting the service level

# Outline

- Motivation ✓
- Idea: Bounded-Time Recovery (BTR) ✓
  - Pros and Cons of BTR
- BTR defined ← NEXT
- Solution sketch
- Summary

# A proposed definition

- Bounded-time recovery:
  - A system offers BTR with a time bound R if its outputs are correct in any interval $[t_1,t_2]$ such that no fault has manifested in $[t_1-R,t_2]$

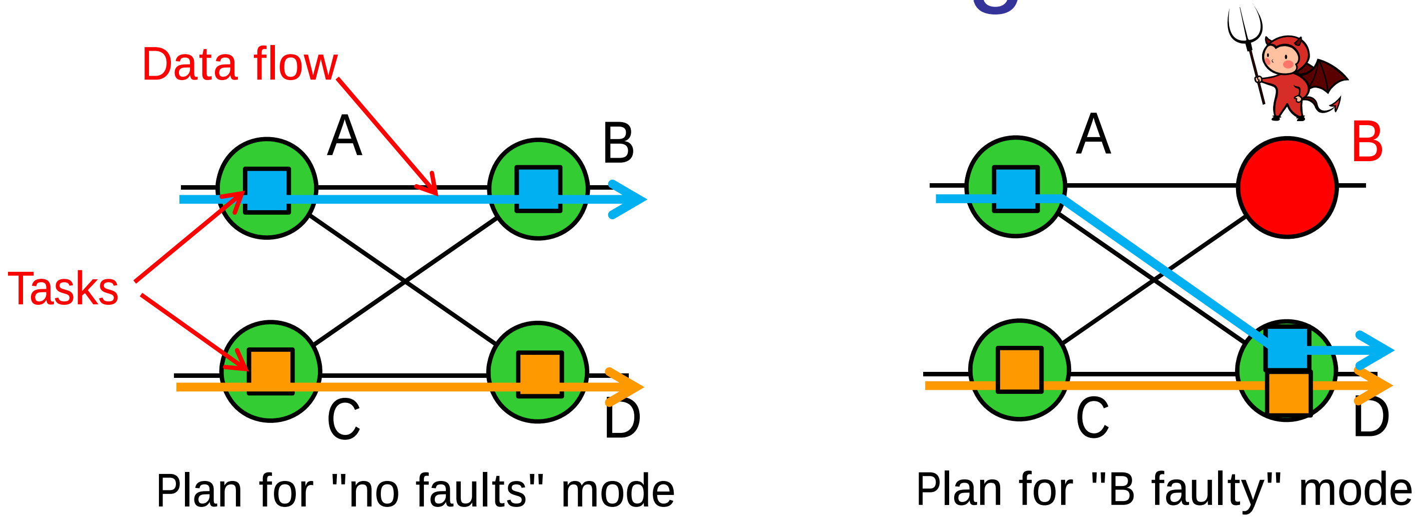



- Some special cases:
  - R=0: Similar to BFT (but with timing guarantees!)
  - $R=\infty$: Similar to self-stabilization
  - Small values of R are the most interesting (and the hardest)

# What assumptions do we need?

- BTR talks about time → Need synchrony!
  - Must have strong bounds on execution times
  - Must have strong bounds on message delays
- This is reasonable (in the CPS domain)
  - WCETs are often known or can be derived
  - Networks have FEC and support bandwidth reservations
- Can we assume Byzantine faults?
  - Real, growing concern for CPS!
  - Qualified yes: Some hardware features needed
    - Example: Protection against Babbling Idiots -- e.g., bus guardians

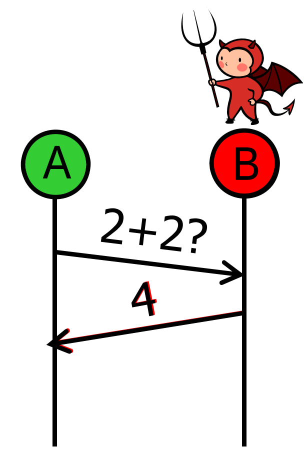
# Solution sketch: Planning




- Ingredient #1: Planner
  - System can run in several modes, has a (static) plan for what to run where in each mode
  - Online vs. offline planning
  - Several interesting challenges (see paper for details)
    - Example: Inter-mode dependencies; connections to game theory
    - Example: Distributed mixed-mode scheduling
  - Interesting opportunites, e.g., fine-grained responses

# Solution sketch: Detection

- Ingredient #2: Fault detector
  - Need to detect (at runtime) when a node misbehaves
  - Can we use PeerReview [SOSP'07] for this?
  - No - PeerReview is for asynchronous systems!



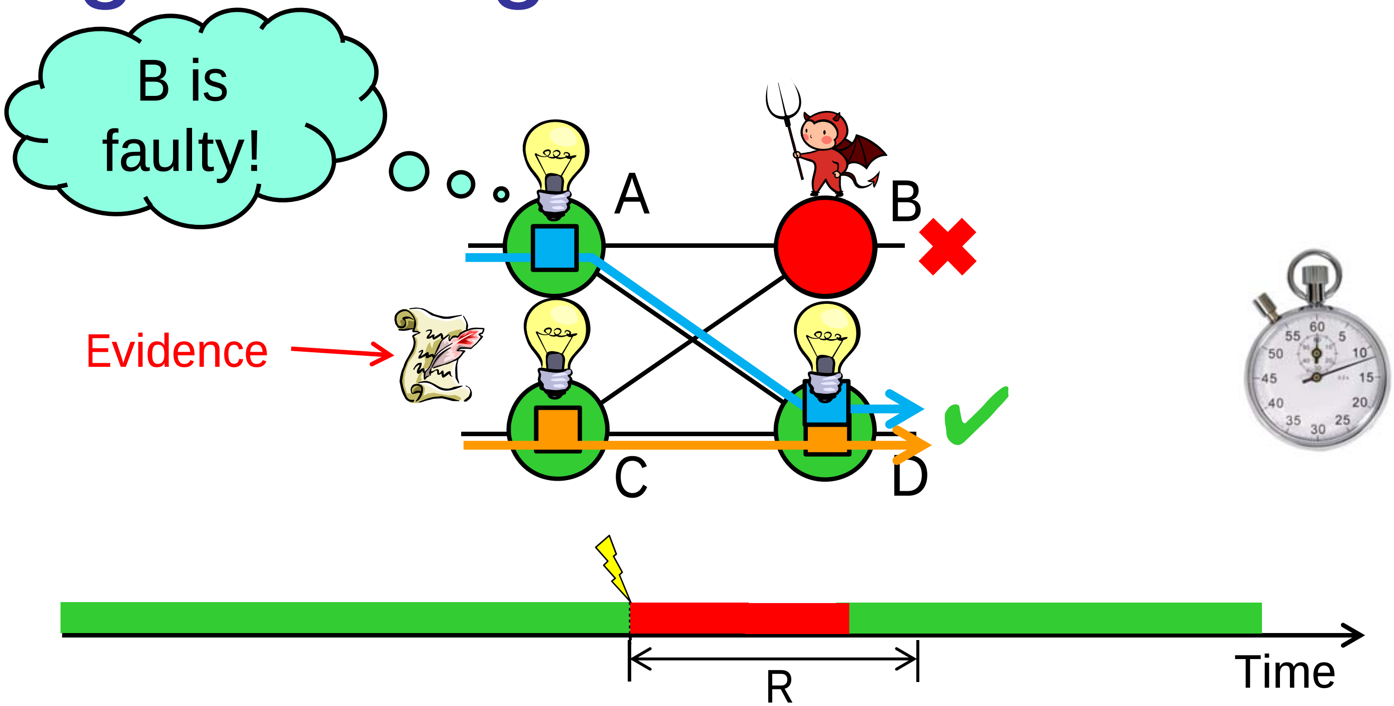

- Challenge: Detecting temporal faults
  - Example: Faulty node might send the right message at the wrong time

- Challenge: Bounding time to detection
  - Adversary can 'win' simply by delaying detection (and thus recovery) for too long!

# Solution sketch: Recovery

- Ingredient #3: Evidence distributor
  - Need to convince other nodes that a fault really exists
    - Adversary might try to confuse the system by reporting non-existent faults
  - PeerReview-style protocols can provide evidence of faults
  - Challenge: Needs resources, new kinds of evidence
- Ingredient #4: Mode switcher
  - Each node needs to switch to the new plan
  - Involves transferring state, starting/terminating tasks
    - Some existing work on mode-change protocols
  - Surprisingly, global agreement may not be needed

# Putting it all together




- Planning: Decide what to run where in each mode
- Detection: Nodes audit each other to look for faults
- Evidence: Nodes prove existence of detected faults
- Mode change: System reconfigures

# Summary

- We propose Bounded-Time Recovery (BTR)
  - New approach to fault tolerance
  - System is allowed to produce wrong outputs after a fault, but only for a limited time
- Case study: Cyber-physical systems
  - Support the additional assumptions that BTR requires
  - BTR could offer lower cost, fine-grained responses to faults
- Interesting research challenges
  - Unusual scheduling problems, new detection protocols, ...

Questions?