Fault Tolerance in Distributed Databases/Systems

Let it crash

How bad it can be ?

Ariane 5 - 4 june 1996

- 10 years of research
- $7 billion invested
- exploded within minute of take off
- Loss estimate $370 million
- but, why ?
  - Someone tried hard to stuff a 64 bit float into a 16 bit int

The nightmare of Wall Street

- $460 million lost in 30 minutes
- why ?
- new version reuses a flag for since long unused feature
- by mistake, 1 of 8 servers is not upgraded to new version
- bad things happen

Thats not an end …

- One of the attempted fixes is to downgrade the servers
- But flag was still set on the servers; now all are faulty
- Bad things happen

Avoid Embarrassing Situations

- Choose right tools for the purpose
- Programming language is also a major contributor
- Consider Replication and failover
- Validation vs Failure
- Failure is not an option
- Failure is a first class citizen in distributed systems. **So, Deal with it**

Traditional RPC

- What if request is lost
- What if response is lost
- Caller is held hostage by the callee

Death & Delay of Distributed Programs

- There is no apparent difference between death and delay in distributed systems
- Distributed programming is all about retries and timeouts
- Without distribution you will always have a SPOF
- More hardware you have higher the failures

Handle Embarrassing Situations

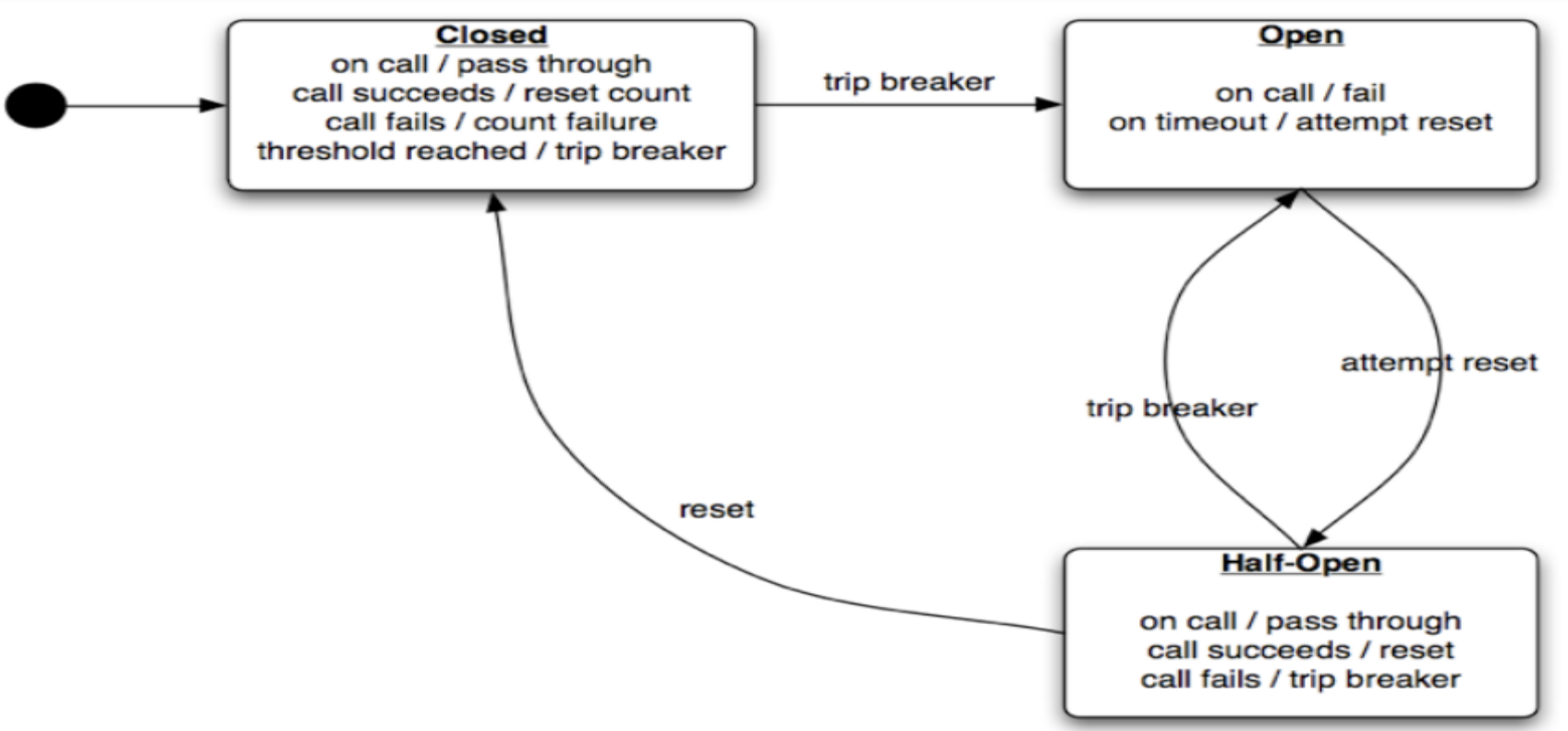
“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.”

- Leslie Lamport

Timeouts

- Thread.wait(timeout)
- ReentrantLock.tryLock
- BlockingQueue.poll(timeout,timeUnit)/offer(..)
- FutureTask.get(timeout, timeUnit)
- Socket.setSoTimeOut(timeout) etc.

Circuit Breaker



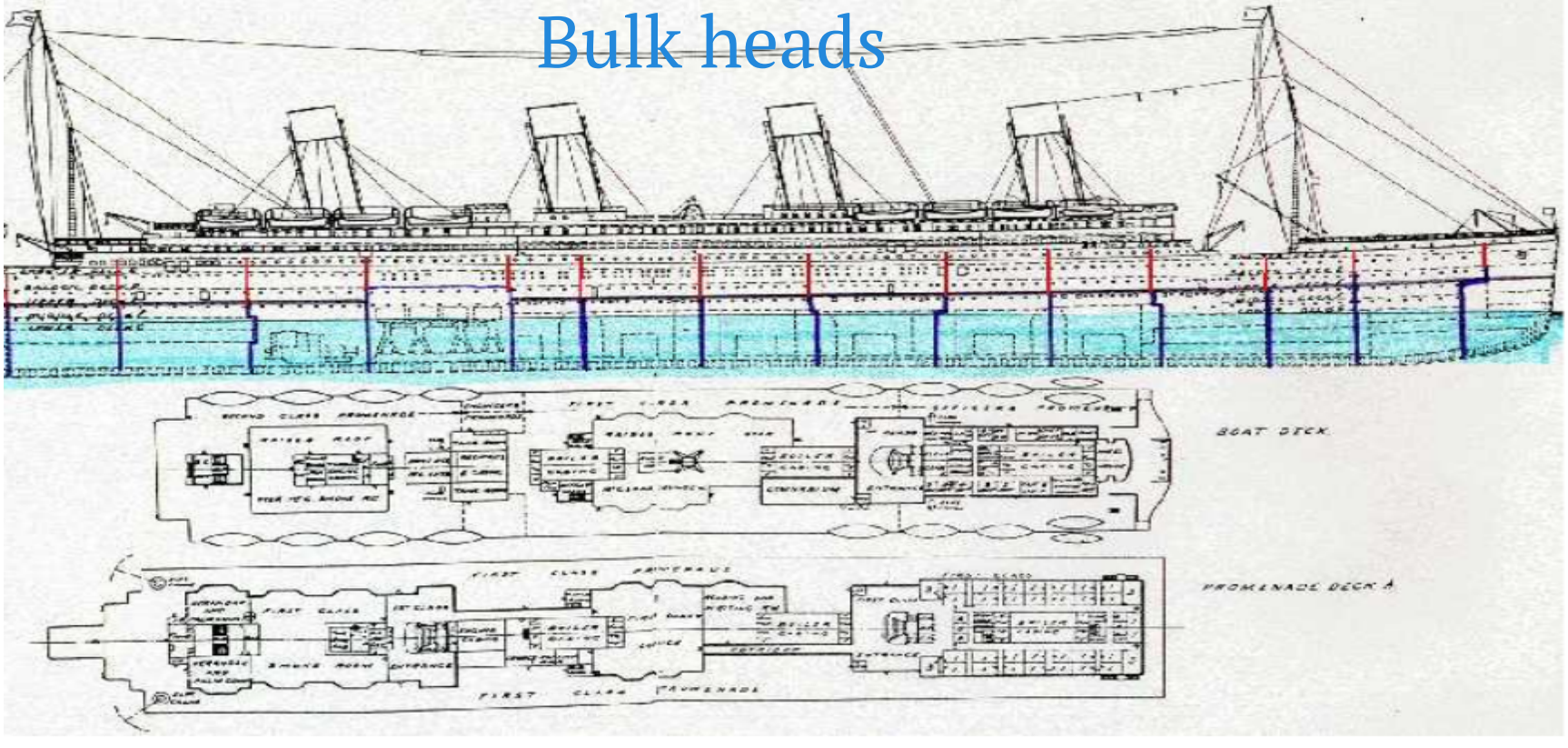
Fail Fast

- Avoid “slow responses”
- Separate:
  - SystemError - resources not available
  - ApplicationError - bad user input etc
- Verify resource availability before starting expensive task
- Input validation immediately

Throttling

- Maintain a steady pace
- Count requests
  - If limit reached, back-off (drop, raise error)
- Used in for example Staged Event-Driven
  - Architecture (SEDA)

Stop Failure Propagation



Bulk heads

- Partition and tolerate failure in one part
- Redundancy
- Applies to threads as well:
  - One pool for admin tasks to be able to perform tasks even though all threads are blocked

Steady State

- Clean up after you doing an side effecting operation.

- Logging:
  - RollingFileAppender (log4j)
  - logrotate (Unix)
  - Scribe - server for aggregating streaming log data
  - Always put logs on separate disk

Self Healing

- Embrace Failure
  - Failure is a normal part of the application lifecycle
- Self Heal
  - Failure is detected, isolated, and managed

CQRS & Event Sourcing

- If I accept a Command and change State
  - Persist Event to Store
- If I crash
  - Replay Events to recover State

Let it Crash Model

- Embrace failure as a natural state in the life-cycle of the application.
- Instead of trying to prevent it; manage it
- Process supervision.
- Supervisor hierarchies (from Erlang)

Summary

- Failure management
  - is not validation
  - need not be boring
  - is not optional
- There are real consequences
  - there are always ways to avoid them


THANK YOU
FOR LISTENING TO THIS
ROCK ON PRESENTATION