Lessons and Predictions from 25 Years of Parallel Data Systems Development

PARALLEL DATA STORAGE WORKSHOP SC11

BRENT WELCH DIRECTOR, ARCHITECTURE

OUTLINE

- **Theme**
  - Architecture for robust distributed systems
  - Code structure
- **Ideas from Sprite**
  - Naming vs I/O
  - Remote Waiting
  - Error Recovery
- **Ideas from Panasas**
  - Distributed System Platform
  - Parallel Declustered Object RAID
- **Open Problems, especially at ExaScale**
  - Getting the Right Answer
  - Fault Handling
  - Auto Tuning
  - Quality of Service

WHAT CUSTOMERS WANT

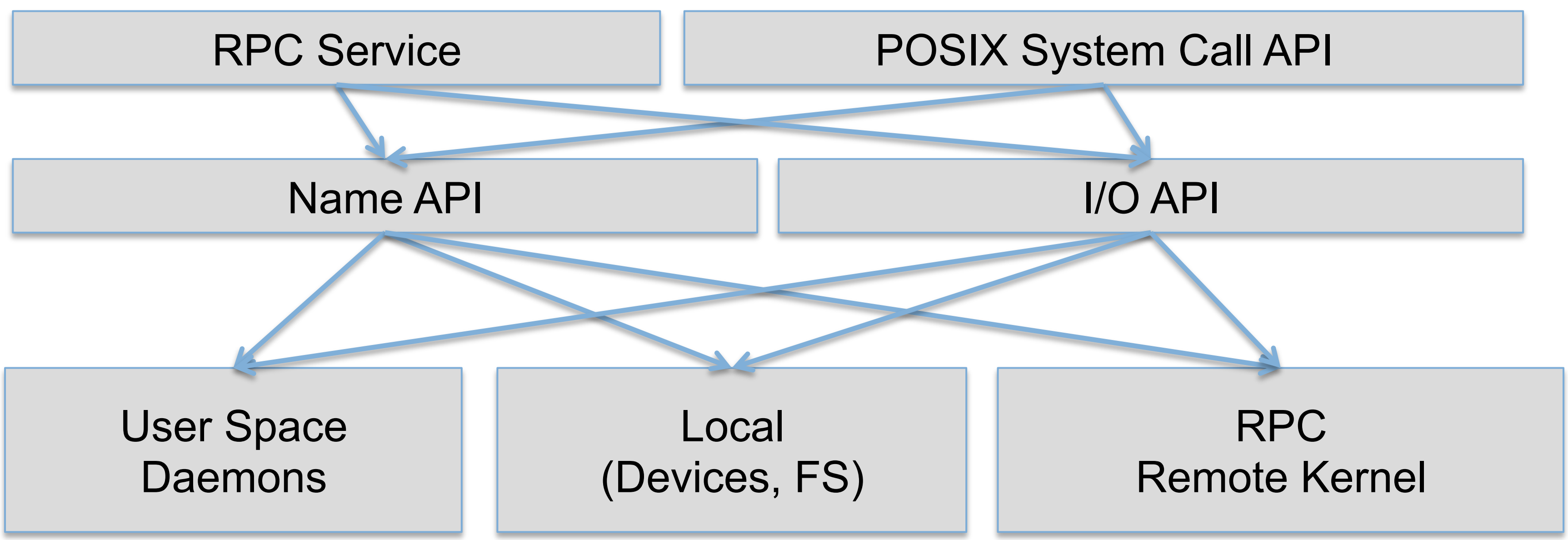
- **Ever Scale, Never Fail, Wire Speed Systems**
  - This is our customer’s expectation
- **How do you build that?**
  - Infrastructure
  - Fault Model

IDEAS FROM SPRITE

- **Sprite OS**
  - UC Berkeley 1984 to 1990's under John Ousterhout
  - Network of diskless workstations and file servers
  - From scratch on Sun2, Sun3, Sun4, DS3100, SPUR hardware
    - 680XX, 8MHz, 4MB, 4-micron, 40MB, 10Mbit/s ("Mega")
  - Supported 5 professors and 25-30 grad student user population
  - 4 to 8 grad students built it. Welch, Fred Douglas, Mike Nelson, Andy Cherenson, Mary Baker, Ken Shirriff, Mendel Rosenblum, John Hartmann
- **Process Migration and a Shared File System**
  - FS cache coherency
  - Write back caching on diskless file system clients
  - Fast parallel make
  - LFS log structured file system
- **A look under the hood**
  - Naming vs I/O
  - Remote Waiting
  - Host Error Monitor

VFS: NAMING VS IO

- **Naming**
  - Create, Open, GetAttr, SetAttr, Delete, Rename, Hardlink
- **I/O**
  - Open, Read, Write, Close, Ioctl
- **3 implementations each API**
  - Local kernel
  - Remote kernel
  - User-level process
- **Compose different naming and I/O cases**



NAMING VS I/O SCENARIOS

panasas

File Server(s)

Names for Devices and Files
Storage for Files

Directory tree is on file servers

Devices are local or on a specific host

Namespace divided by prefix tables

User-space daemons do either/both API

Diskless Node

Local Devices

`/dev/console`

`/dev/keyboard`

Special Node

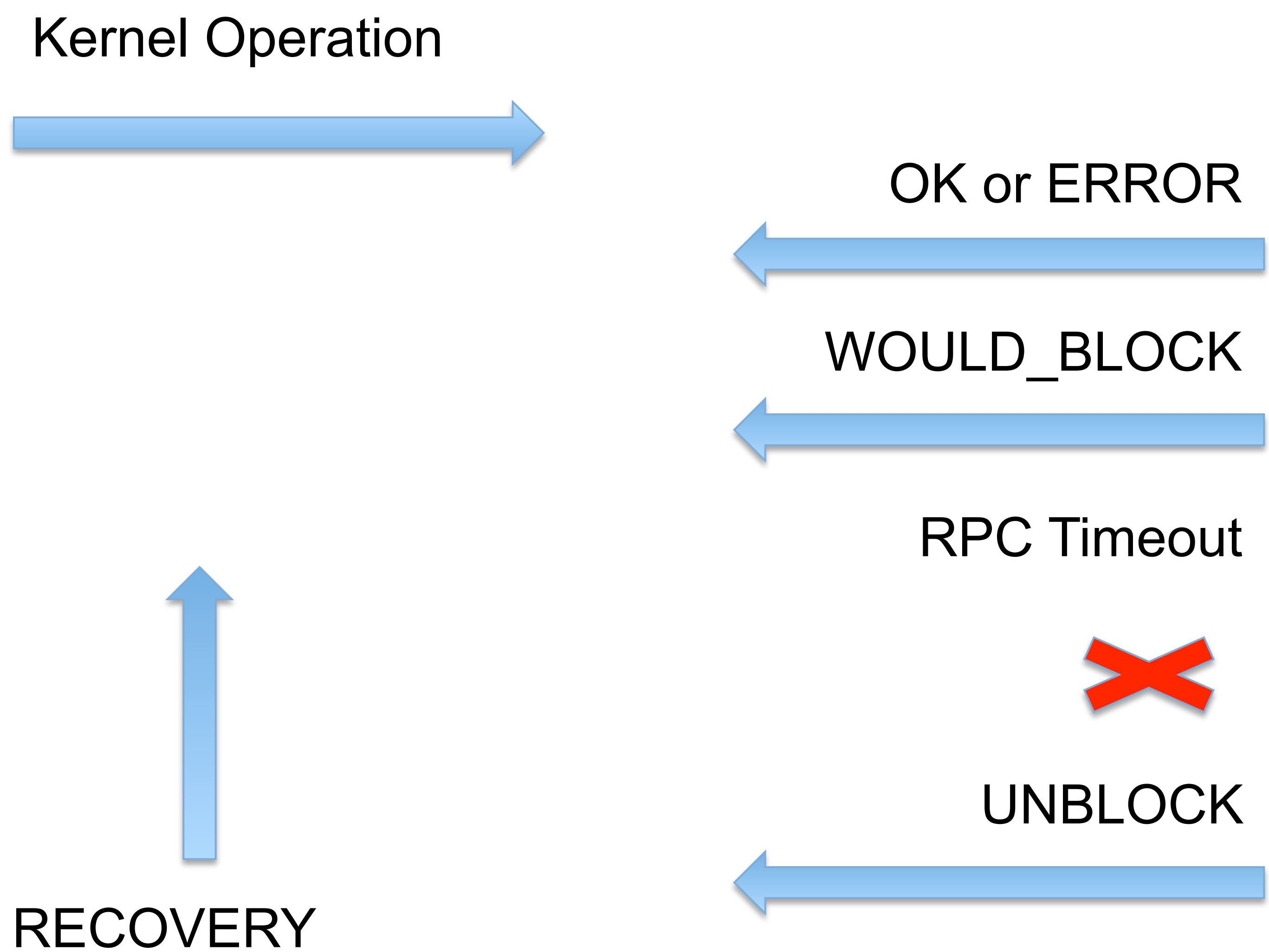
Shared Devices

`/host/allspice/dev/tape`

User Space Daemon

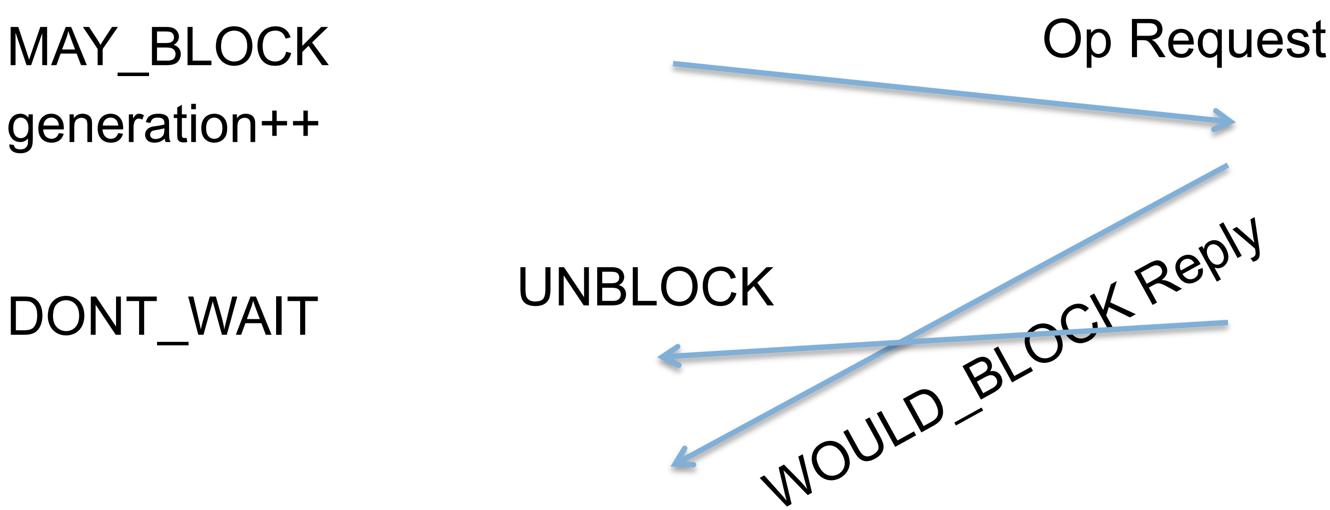
`/tcp/ipaddr/port`

SPRITE FAULT MODEL



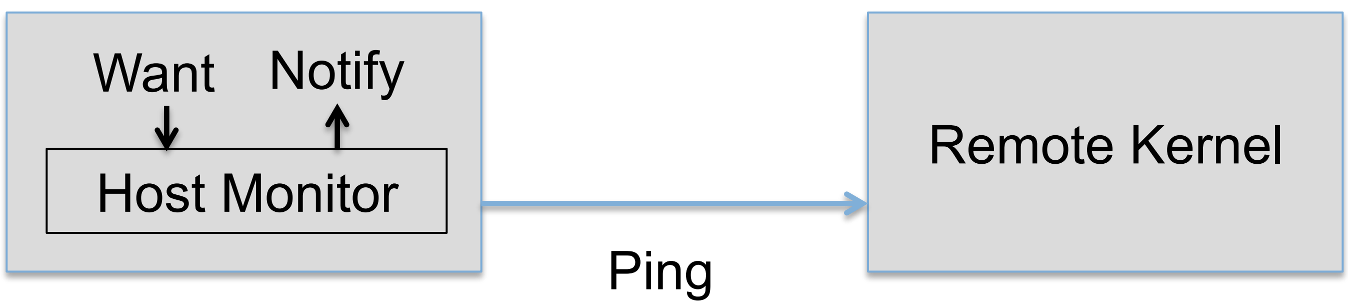
REMOTE WAITING

- **Classic Race**
  - WOULD_BLOCK reply races with UNBLOCK message
  - Race ignores unblock and request waits forever
- **Fix: 2-bits and a generation ID**
  - Process table has “MAY_BLOCK” and “DONT_WAIT” flag bits
  - Wait generation ID incremented when MAY_BLOCK is set
  - DONT_WAIT flag is set when race is detected based on generation ID



HOST ERROR MONITOR

- **API: Want Recovery, Wait for Recovery, Recovery Notify**
  - Subsystems register for errors
  - High-level (syscall) layer waits for error recovery
- **Host Monitor**
  - Pings remote peers that need recovery
  - Triggers Notify callback when peer is ready
  - Makes all processes runnable after notify callbacks complete



SPRITE SYSTEM CALL STRUCTURE

- **System call layer handles blocking conditions, above VFS API**

```
Fs_Read(streamPtr, buffer, offset, lenPtr) {
  setup parameters in ioPtr
  while (TRUE) {
    Sync_GetWaitToken(&waiter);
    rc = (fsio_StreamOpTable[streamType].read)
         (streamPtr, ioPtr, &waiter, &reply);
    if (rc == FS_WOULD_BLOCK) {
         rc = Sync_ProcWait(&waiter);
    }
    if (rc == RPC_TIMEOUT || rc == FS_STALE_HANDLE ||
                        rc == RPC_SERVICE_DISABLED)  {
         rc = Fsutil_WaitForRecovery(streamPtr->ioHandlePtr, rc);
    }
    break or continue as appropriate
}
```

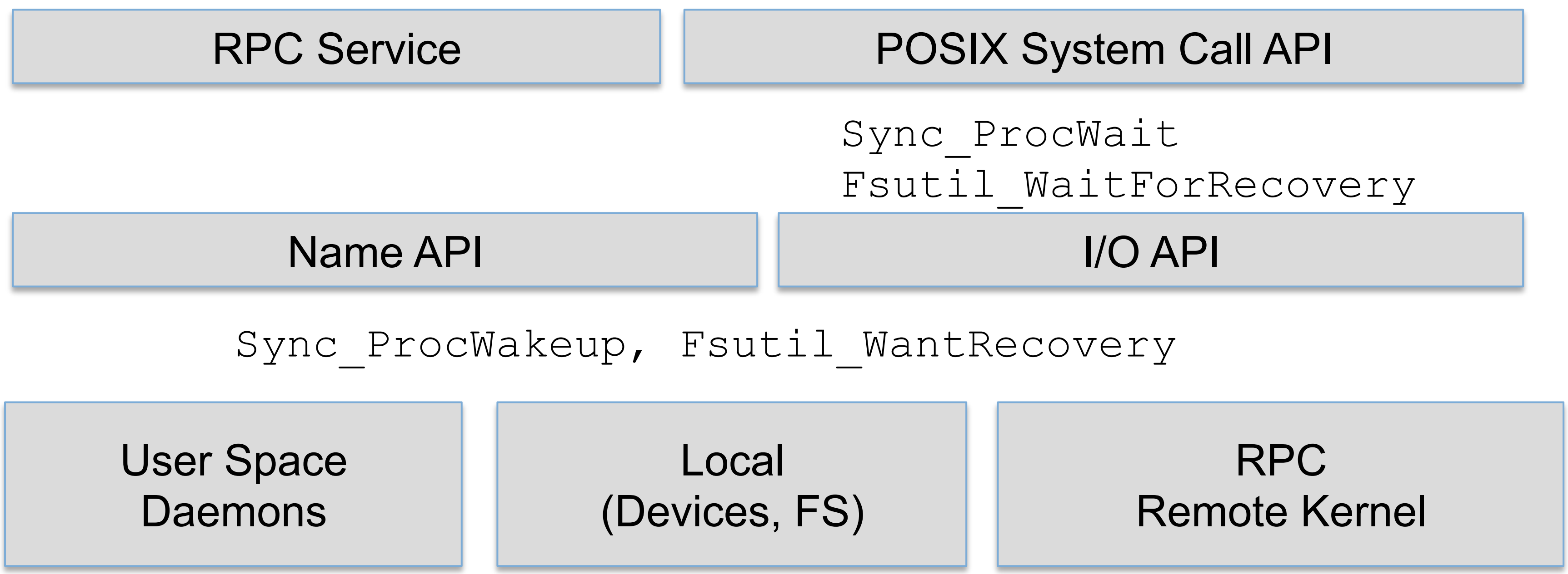
SPRITE REMOTE ACCESS

- **Remote kernel access uses RPC and must handle errors**

```
Fsrmt_Read(streamPtr, ioPtr, waitPtr, replyPtr) {
  loop over chunks of the buffer {
    rc = Rpc_Call(handle, RPC_FS_READ, parameter_block);
    if (rc == OK || rc == FS_WOULD_BLOCK) {
        update chunk pointers
        continue, or break on short read or FS_WOULD_BLOCK
    } else if (rc == RPC_TIMEOUT) {
        rc = Fsutil_WantRecovery(handle);
        break;
    }
    if (done) break;
  }
  return rc;
}
```

SPRITE ERROR RETRY LOGIC

- **System Call Layer**
  - Sets up to prevent races
  - Tries an operation
  - Waits for blocking I/O or error recovery w/out locks held
- **Subsystem**
  - Takes Locks
  - Detects errors and registers the problem
  - Reacts to recovery trigger
  - Notifies waiters

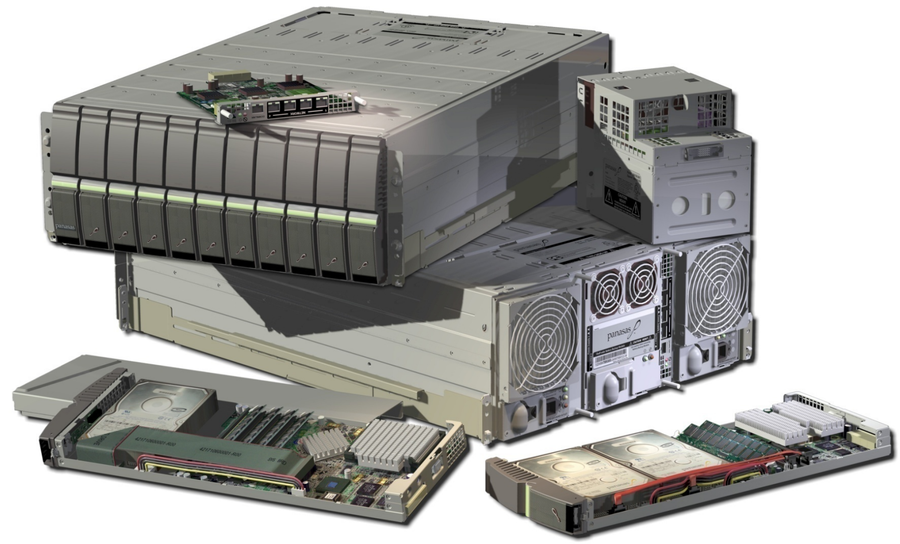


SPRITE

- **Tightly coupled collection of OS instances**
  - Global process ID space (host+pid)
  - Remote wakeup
  - Process migration
  - Host monitor and state recovery protocols
- **Thin “Remote” layer optimized by write-back file caching**
  - General composition of the remote case with kernel and user services
  - Simple, unified error handling

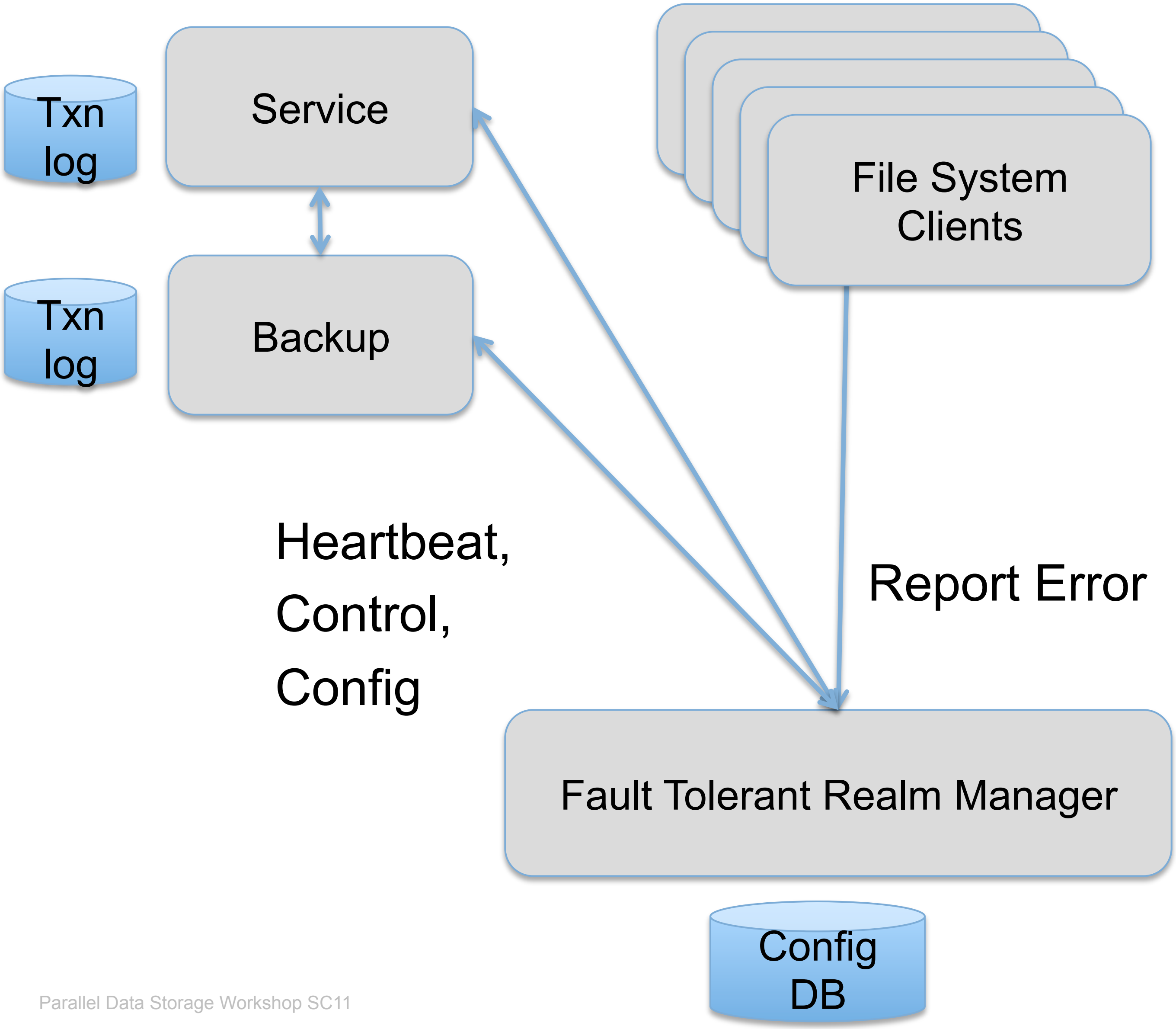
IDEAS FROM PANASAS

- **Panasas Parallel File System**
  - Founded by Garth Gibson
  - 1999-2011+
  - Commercial
  - Object RAID
  - Blade Hardware
  - Linux RPM to mount /panfs



- **Features**
  - Parallel I/O, NFS, CIFS, Snapshots, Management GUI, *Hardware/ Software fault tolerance*, Data Management APIs
- **Distributed System Platform**
  - Lamport's PAXOS algorithm
- **Object RAID**
  - NASD heritage

PANASAS FAULT MODEL

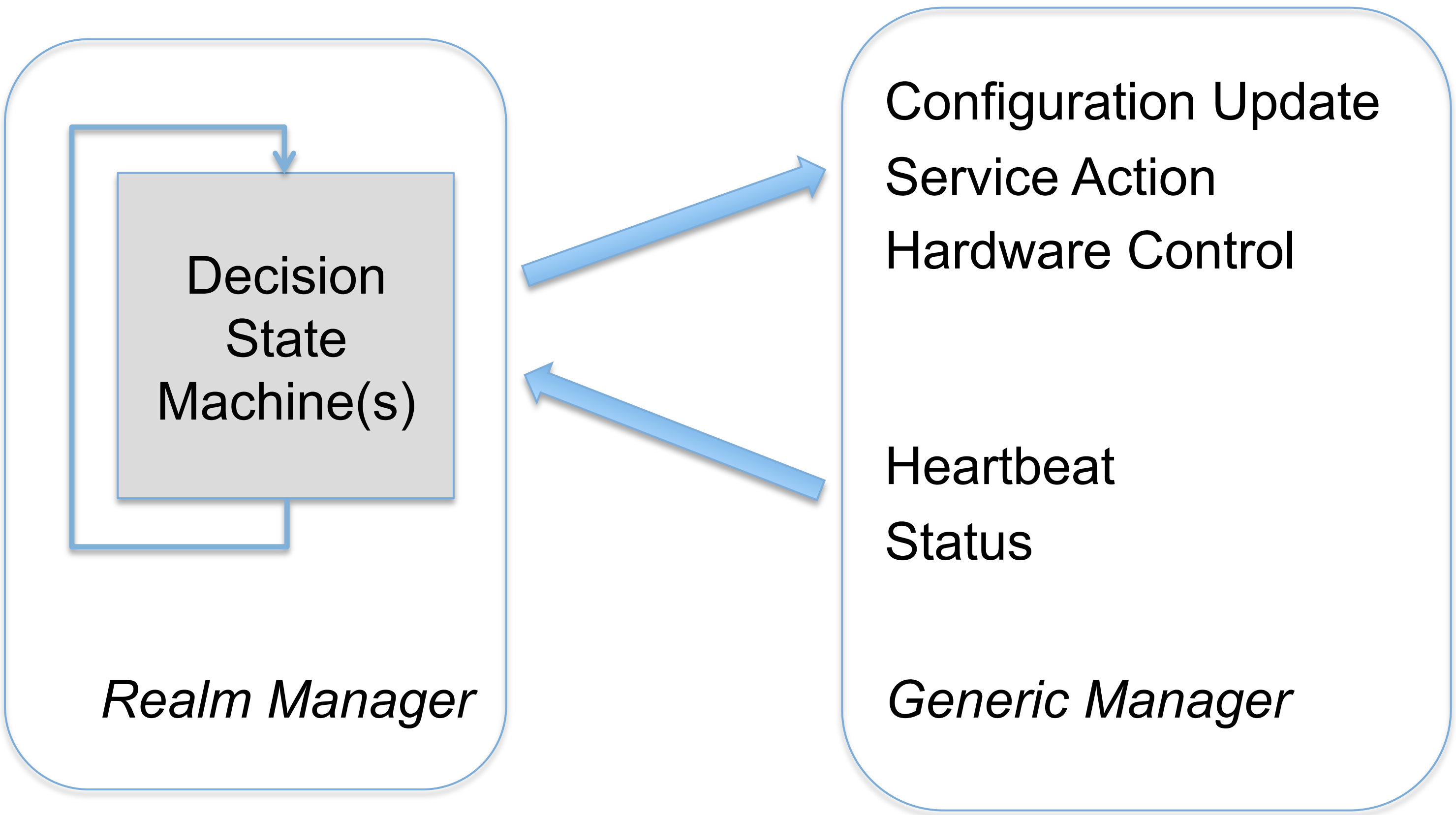


PANASAS DISTRIBUTED SYSTEM PLATFORM

- ***Problem*: managing large numbers of hardware and software components in a highly available system**
  - What is the system configuration?
  - What hardware elements are active in the system?
  - What software services are available?
  - What software services are activated, or backup?
  - What is the desired state of the system?
  - What components are failed?
  - What recovery actions are in progress?
- ***Solution*: Fault-tolerant Realm Manager to control all other software services and (indirectly) hardware modules.**
  - Distributed file system one of several services managed by the RM
    - Configuration management
    - Software upgrade
    - Failure Detection
    - GUI/CLI management
    - Hardware monitoring

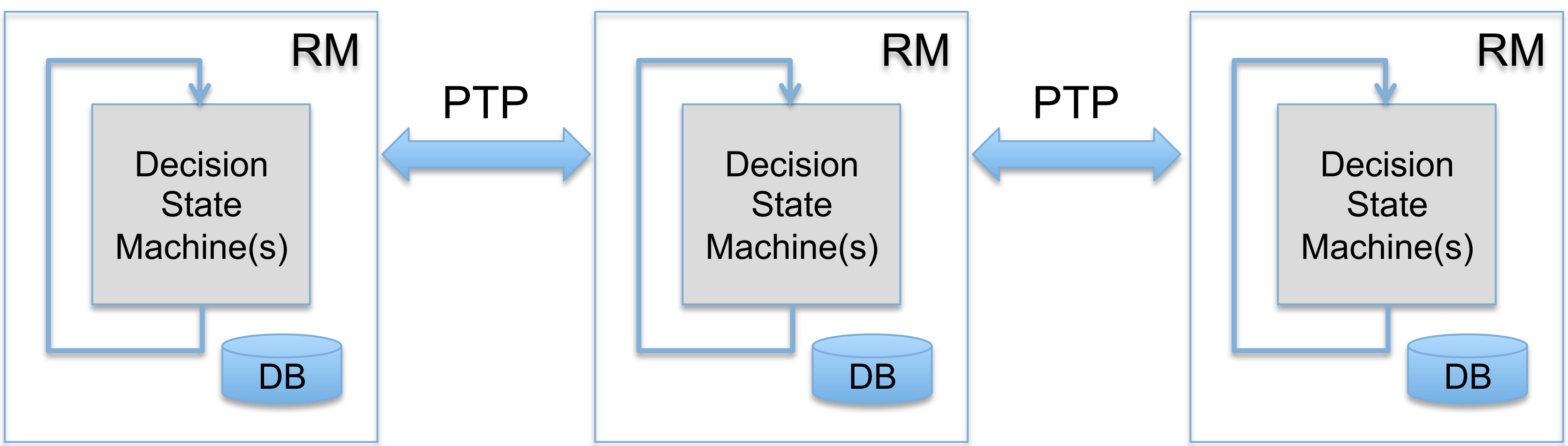
MANAGING SERVICES

- **Control Strategy**
  - Monitor -> Decide -> Control -> Monitor
  - Controls act on *<u>one or more</u>* distributed system elements that can fail
  - State Machines have “Sweeper” tasks to drive them periodically



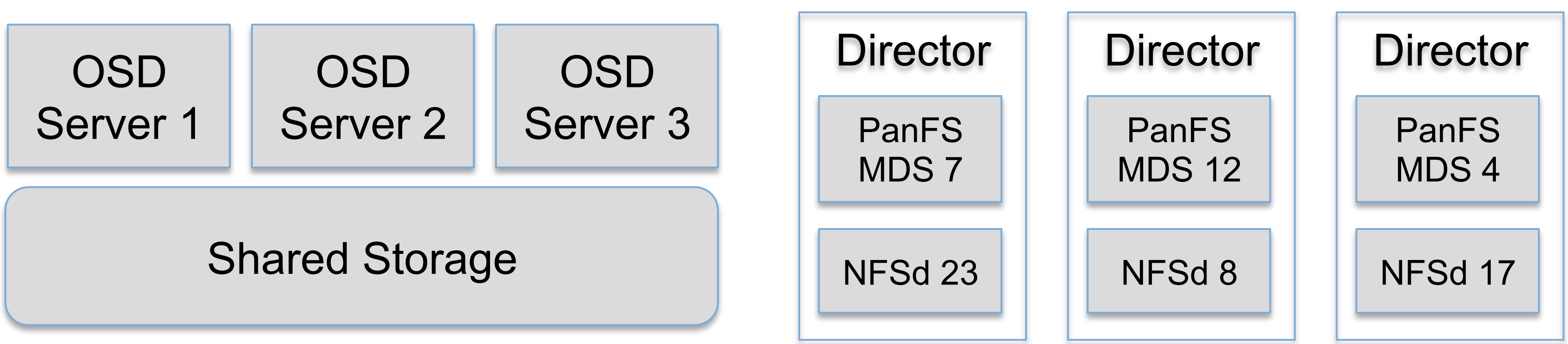
FAULT TOLERANT REALM MANAGER

- **PTP Voting Protocol**
  - 3-way or 5-way redundant Realm Manager (RM) service
  - PTP (Paxos) Voting protocol among majority quorum to update state
- **Database**
  - Synchronized state maintained in a database on each Realm Manager
  - State machines record necessary state persistently
- **Recovery**
  - Realm Manager instances fail stop w/out a majority quorum
  - Replay DB updates to re-joining members, or to new members



LEVERAGING VOTING PROTOCOLS (PTP)

- **Interesting activities require multiple PTP steps**
  - Decide – Control – Monitor
  - Many different state machines with PTP steps for different product features
    - Panasas metadata services: primary and backup instances
    - NFS virtual server fail over (pools of IP addresses that migrate)
    - Storage server failover in front of shared storage devices
    - Overall realm control (reboot, upgrade, power down, etc.)
- **Too heavy-weight for file system metadata or I/O**
  - Record service and hardware configuration and status
  - Don't use for open, close, read, write

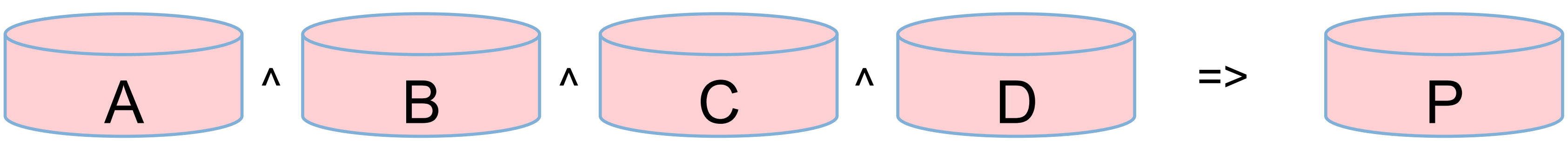


PANASAS DATA INTEGRITY

- **Object RAID**
  - Horizontal, declustered striping with redundant data on different OSDs
  - Per-file RAID equation allows multiple layouts
    - Small files are mirrored RAID-1
    - Large files are RAID-5 or RAID-10
    - Very large files use two level striping scheme to counter network incast
- **Vertical Parity**
  - RAID across sectors to catch silent data corruption
  - Repair single sector media defects
- **Network Parity**
  - Read back per-file parity to achieve true end-to-end data integrity
- **Background scrubbing**
  - Media, RAID equations, distributed file system attributes

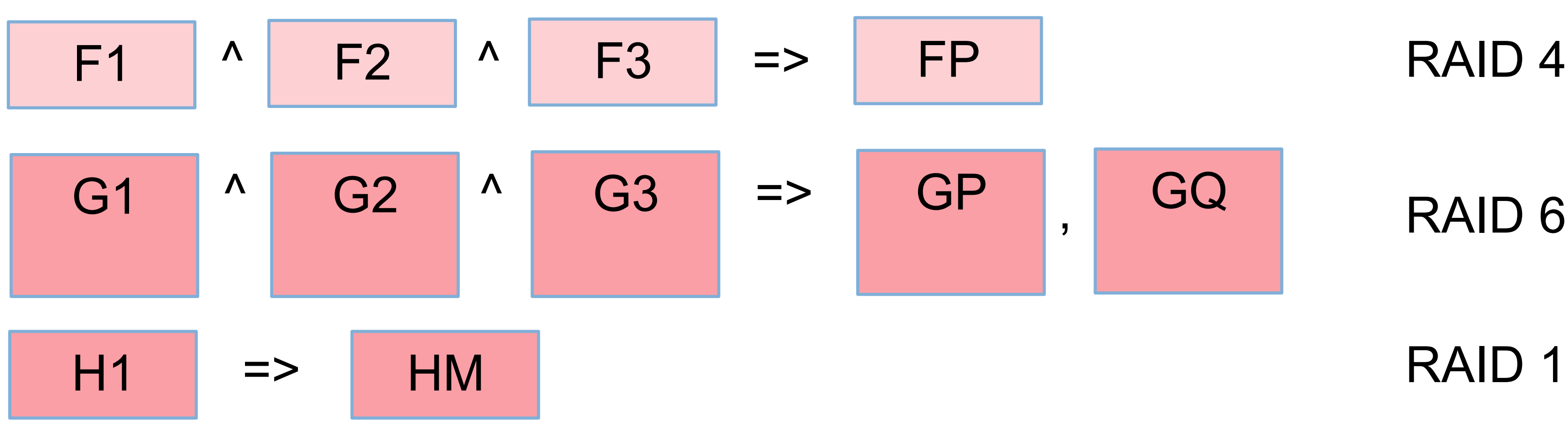
RAID AND DATA PROTECTION

- **RAID was invented for performance (striping data across many slow disks) and reliability (recover failed disk)**
  - RAID equation generates redundant data:
  - P = A xor B xor C xor D (encoding)
  - B = P xor A xor C xor D (data recovery)
- **Block RAID protects an entire disk**



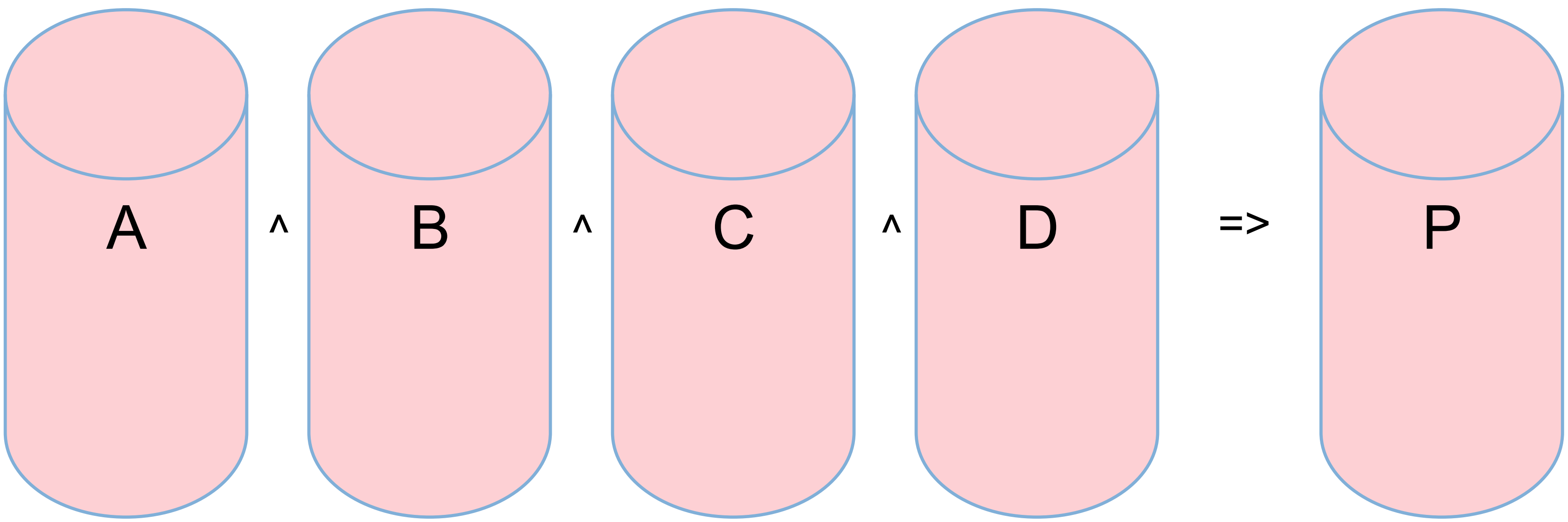
OBJECT RAID

- **Object RAID protects and rebuilds files**
  - Failure domain is a file, which is typically much much smaller than the physical storage devices
  - File writer is responsible for generating redundant data, which avoids central RAID controller bottleneck and <u>*allows end-to-end checkng*</u>
  - Different files sharing same devices can have different RAID configurations to vary their level of data protection and performance



THE PROBLEM WITH BLOCK RAID

- **Traditional block-oriented RAID protects and rebuilds entire drives**
  - Unfortunately, *drive capacity increases have outpaced drive bandwidth*
  - It takes longer to rebuild each new generation of drives
  - Media defects on surviving drives interfere with rebuilds



BLADE CAPACITY AND SPEED HISTORY

Compare time to write a blade (two disks) from end-to-end over 4* generations of Panasas blades

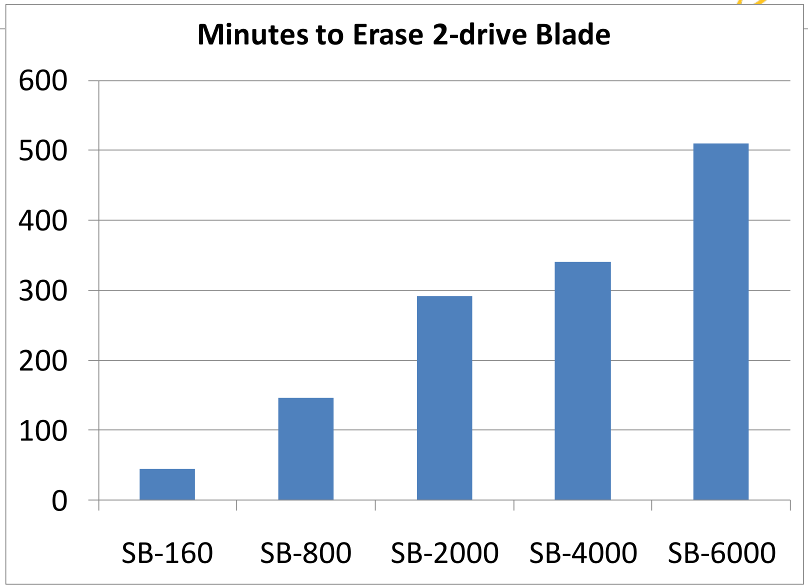
SB-4000 same family as SB-6000

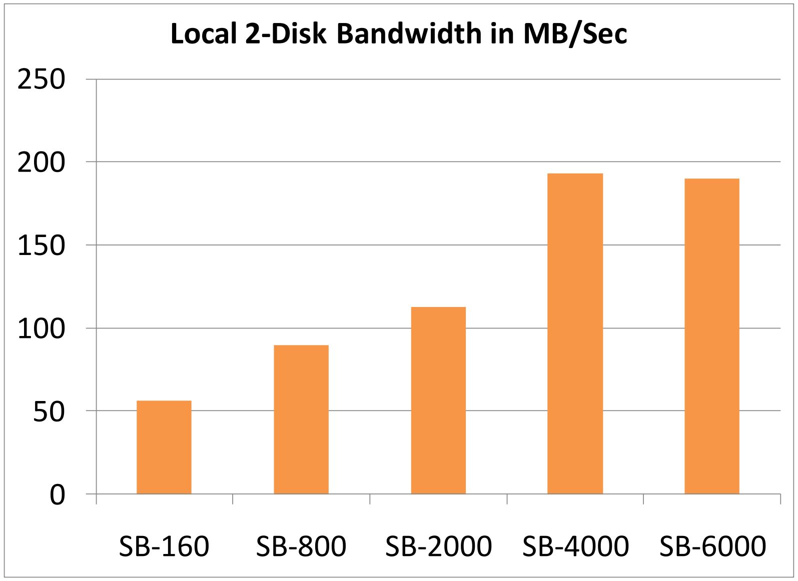
Capacity increased 39x

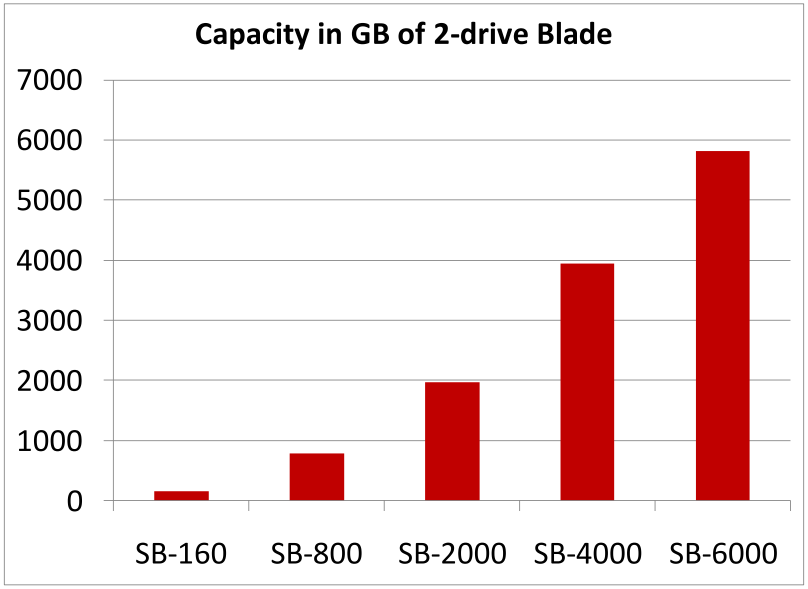
Bandwidth increased 3.4x

(function of CPU, memory, disk)

Time goes from 44 min to > 8 hrs

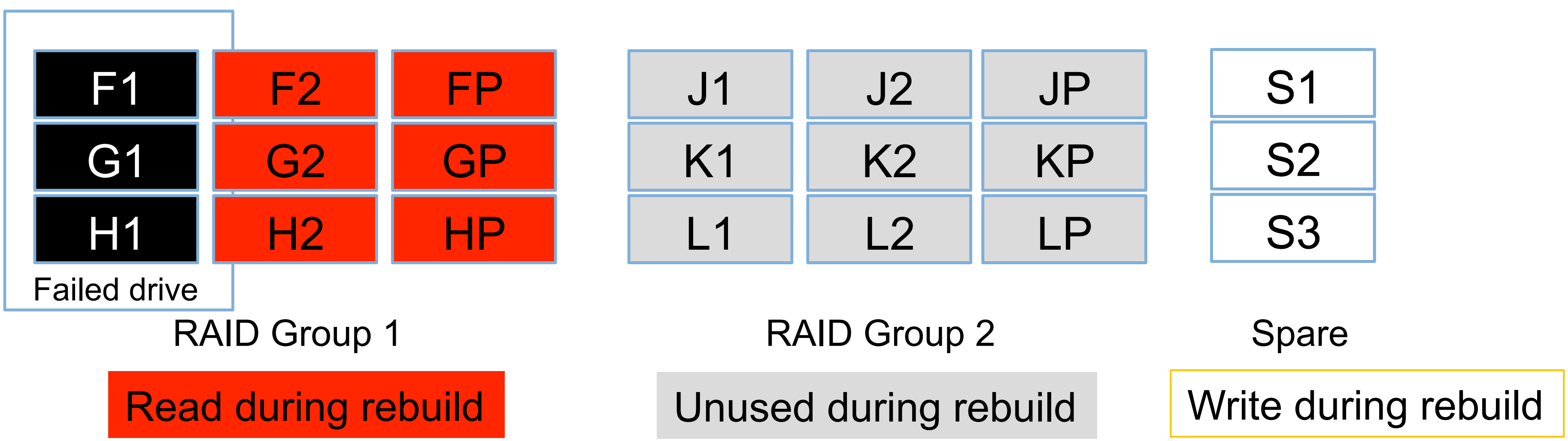






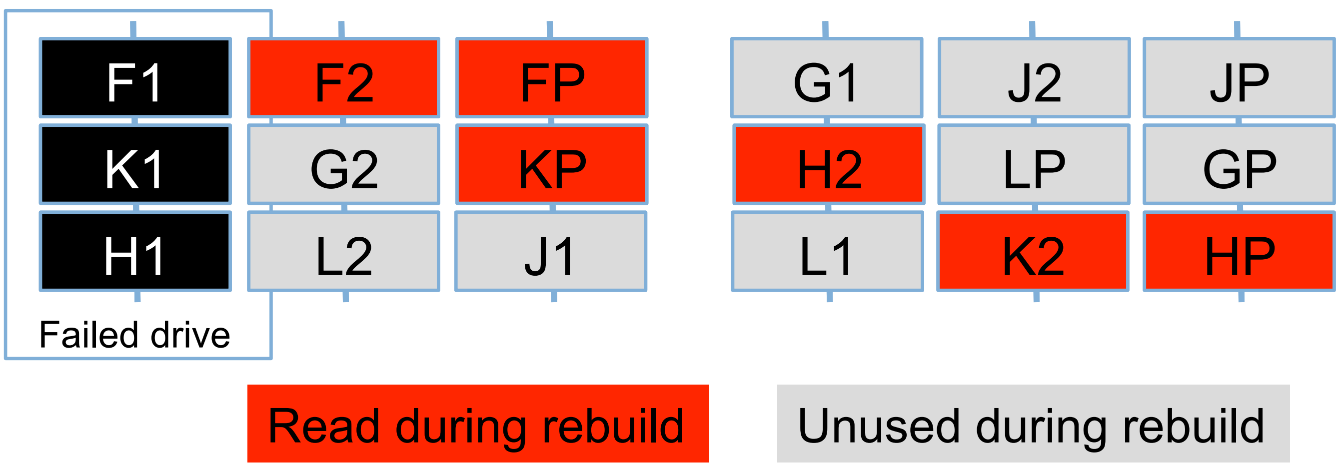
TRADITIONAL RAID REBUILD

- **RAID requires I/O bandwidth, memory bandwidth and CPU**
  - Rebuilding a 1TB drive in a 5-drive RAID group reads 4TB and writes 1TB
    - RAID-6 rebuilds after two failures require more computation and I/O
  - *Rebuild workload creates hotspots*
    - Parallel user workloads need uniform access to all spindles
- **Example: 2+1 RAID, 6 Drives, 2 Groups, 1 Spare Drive**



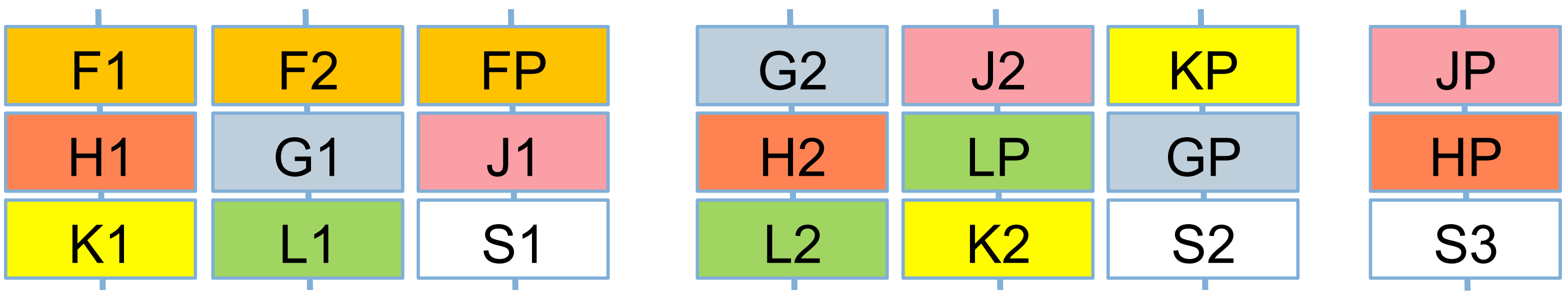
DECLUSTERED DATA PLACEMENT

- **Declustered placement uses the I/O bandwidth of many drives**
  - Declustering spreads RAID groups over larger number of drives to amplify the disk and network I/O available to the RAID engines
  - 2 Disks of data read from 1/3 or 2/3 of 5 remaining drives
    - With more placement groups (e.g., 100), finer grain load distribution
- **Example: 2+1 RAID, 6 Drives, 6 Groups**



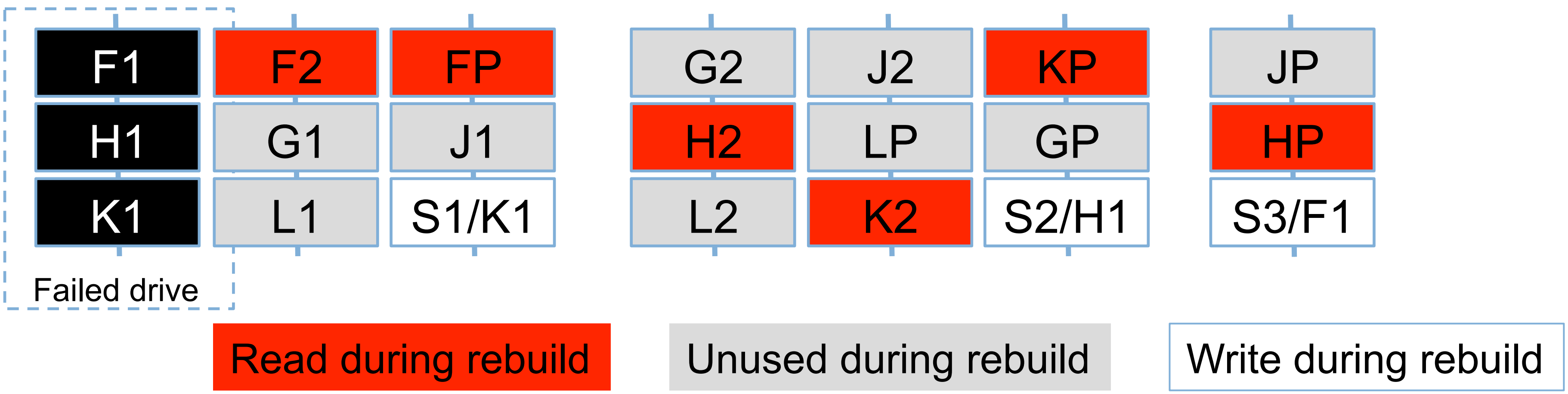
DECLUSTERED SPARE SPACE

- **Declustered spare space improves write I/O bandwidth**
  - 1 Disk of data written to 1/3 of 2 or 3 remaining drives
- **Spare location places constraints that must be honored**
  - Cannot rebuild onto a disk with another element of your group
- **Example: 2+1 RAID, 7 Drives, 6 Groups, 1 Spare**



PARALLEL DECLUSTERED RAID REBUILD

- **Parallel algorithms harness the power of many computers, and for RAID rebuild, the I/O bandwidth of many drives**
  - Group rebuild work can be distributed to multiple “RAID engines” that have access to the data over a network
    - Scheduler task supervises worker tasks that do group rebuilds in parallel
  - Optimal placement is a hard problem (see Mark Holland, ‘98)
    - Example reads 1/3 of each remaining drive, writes 1/3 to half of them

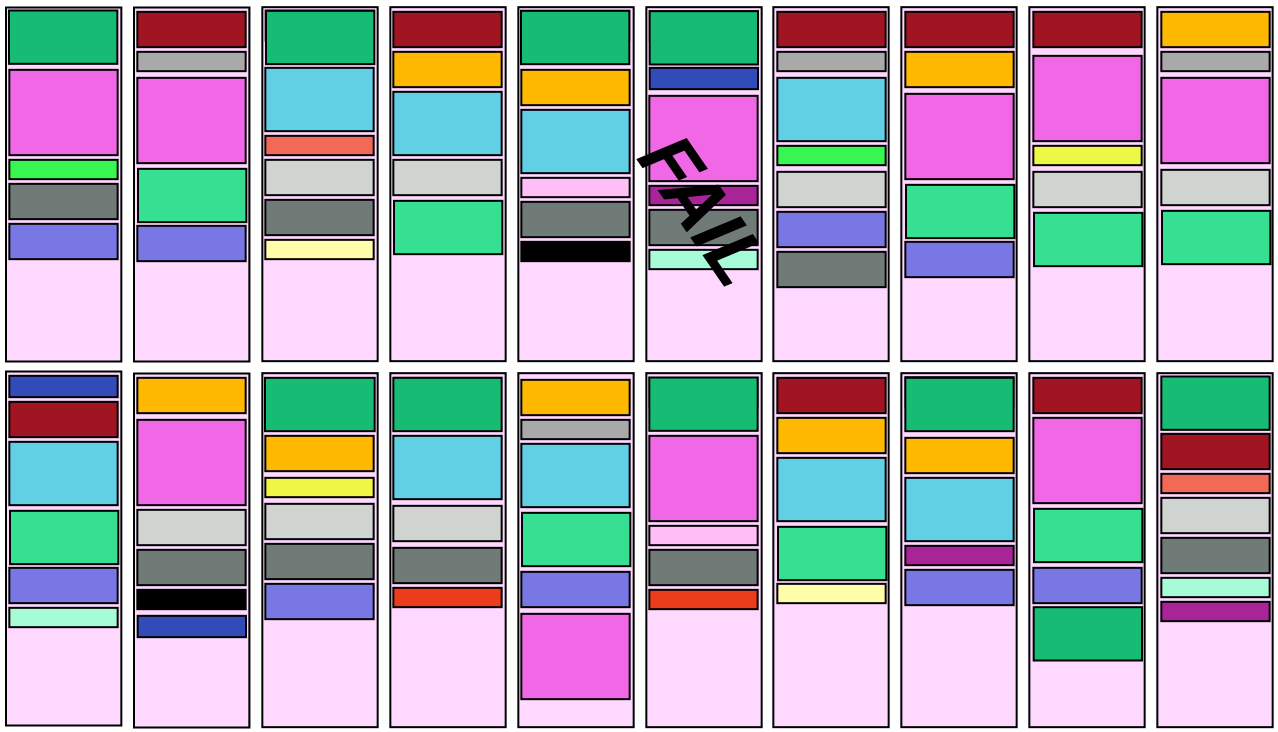


PARALLEL DECLUSTERED OBJECT RAID

- File attributes replicated on first two component objects
- Component objects include file data and file parity
- Components grow & new components created as data written
- Per-file RAID equation creates fine-grain work items for rebuilds
- Declustered, randomized placement distributes RAID workload

20 OSD Storage Pool

Mirrored or 9-OSD Parity Stripes



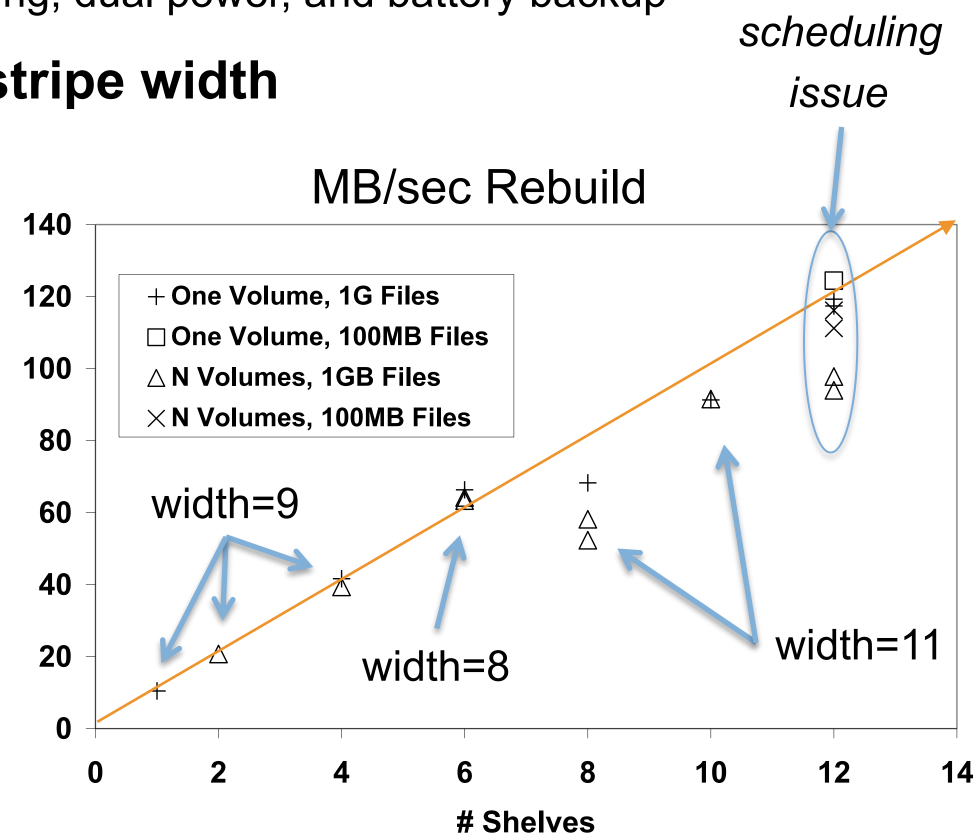
Read about half of each surviving OSD

Write a little to each OSD

Scales up in larger Storage Pools

PANASAS SCALABLE REBUILD

- **RAID rebuild rate increases with storage pool size**
  - Compare rebuild rates as the system size increases
  - Unit of growth is an 11-blade Panasas "shelf"
    - 4-u blade chassis with networking, dual power, and battery backup
- **System automatically picks stripe width**
  - 8 to 11 blade wide parity group
    - Wider stripes slower
  - Multiple parity groups
    - Large files
- **Per-shelf rate scales**
  - 10 MB/s (old hardware)
    - Reading at 70-90 MB/sec
    - Depends on stripe width
  - 30-40 MB/sec (current)
    - Reading at 200-300 MB/sec



HARD PROBLEMS FOR TOMORROW

- **Issues for Exascale**
  - Millions of cores
  - TB/sec bandwidth
  - Exabytes of storage
  - Thousands and Thousands of hardware components
- **Getting the Right Answer**
- **Fault Handling**
- **Auto Tuning**
- **Quality of Service**
- *Better/Newer devices*

GETTING THE RIGHT ANSWER

- **Verifying system behavior *<u>in all error cases</u>* will be very difficult**
  - Are applications computing the right answer?
  - Is the storage system storing the right data?
  - Suppose I know the answer is wrong – what broke?
  - There may be no other computer on the planet capable of checking
  - It may or may not be feasible to prove correctness
- **The test framework should be at least as complicated as the system under test**

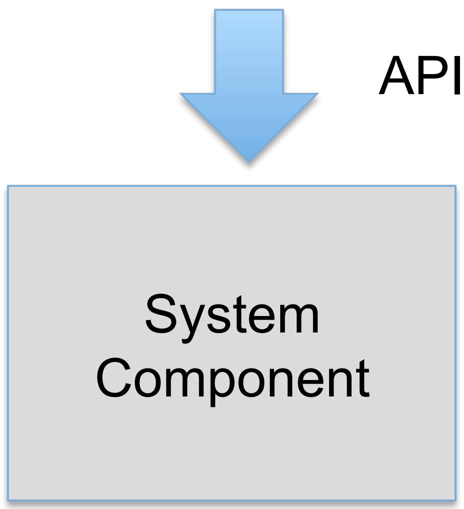
  Bert Sutherland

PROGRAMS THAT RUN FOREVER

- **Ever Scale, Never Fail, Wire Speed Systems**
  - This is our customer's expectation
- **If you can keep it stable as it grows, performance follows**
  - Stability adds overhead
- **Humans and the system need to know what is wrong**
  - Trouble shooting and auto correction will be critical features

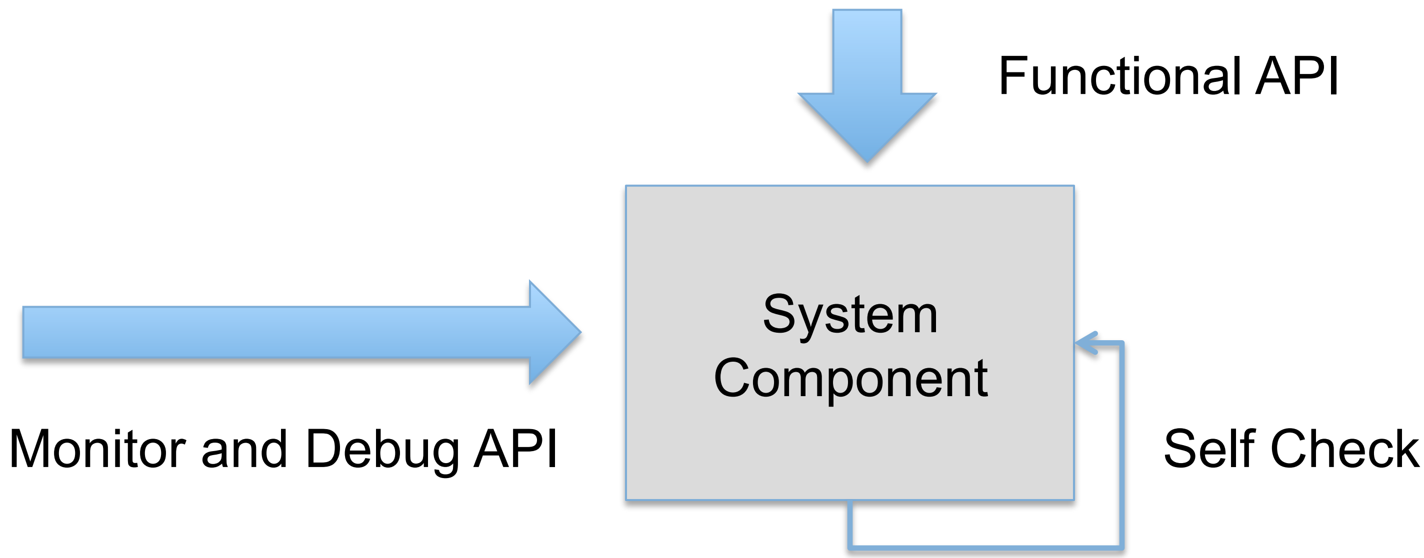
Is it functioning correctly?

What's wrong in the system?



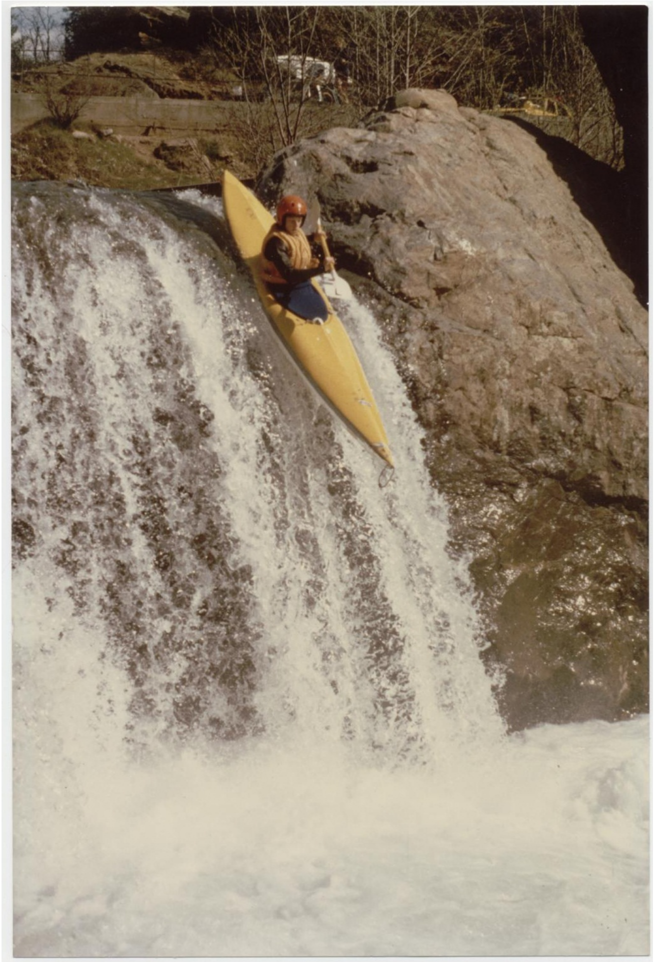
RUGGED COMPONENTS

- **Functional API**
  - Comes from customer requirements
- **Monitor, Debug API**
  - Comes from testing and validation requirements
- **Self Checking**
  - E.g., phone switch “audit” code keeps switches from failing



OBVIOUS STRATEGIES

- **Self checking components that isolate errors**
  - Protocol checksums and message digests
- **Self correcting components that mask errors**
  - RAID, checkpoints, Realm Manager
  - Application-level schemes
    - map-reduce replay of lost work items
- **End-to-end checking**
  - Overall back-stop
  - Application-generated checksums



WHAT ABOUT PERFORMANCE?

- **QoS and Self-Tuning will grow in importance**
  - QoS is a form of self-checking and self-correcting systems
  - How do you provide QoS w/out introducing bottlenecks?
- **Parallel batch jobs crush their competition**
  - E.g., your “ls” or “tar xf” will starve behind the 100,000 core job
- **Stragglers hurt parallel jobs**
  - Why do some ranks run much more slowly than others?
    - Compounded performance bias w/ lack of control system
- **The storage system needs self-awareness and control mechanisms to help these problem scenarios**
  - Open, close, read, write is the easy part
  - Your contributions will be on error handling and control systems

THANK YOU
WELCH@PANASAS.COM