Preliminary Investigations into a Microkernel OSAL for cFS

Gregor Peach, Joseph Espy, Zach Day,
Gabriel Parmer, Alex Maloney
Gerald Fry*, Curt Wu*

The George Washington University
* Charles River Analytics

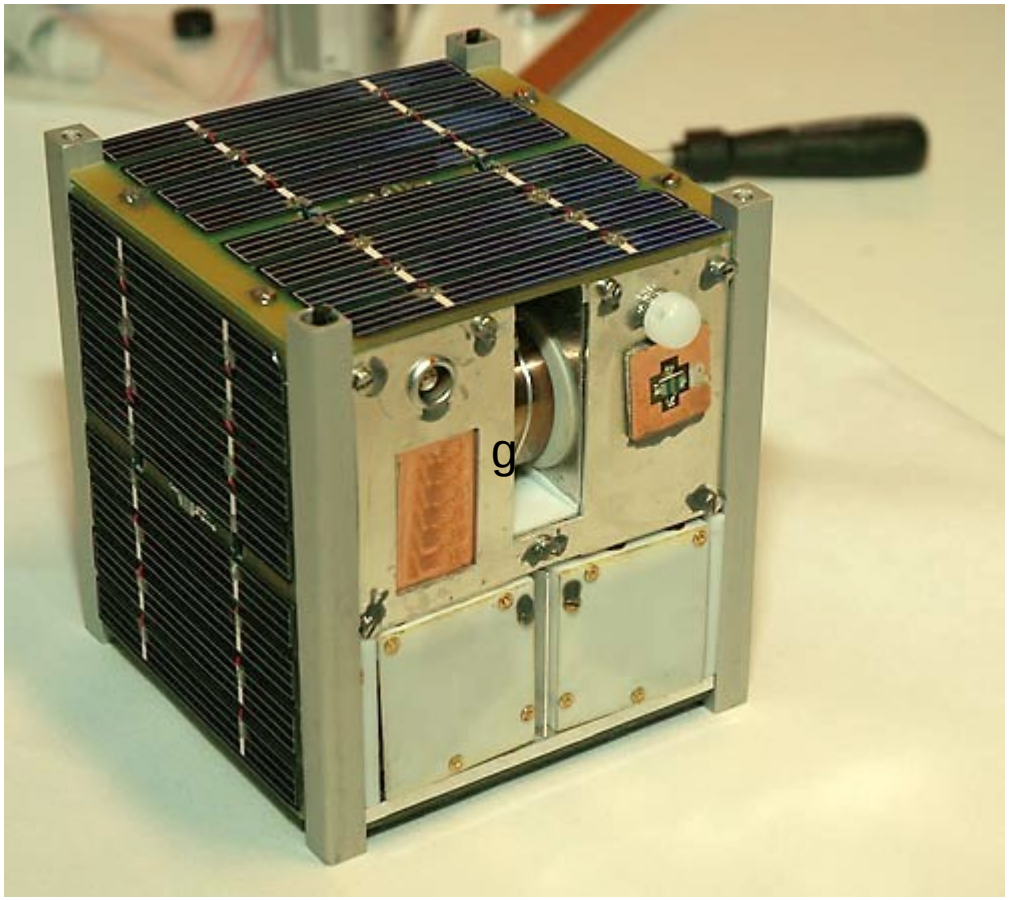
Acknowledgements: This material is based upon work supported by the National Science Foundation under Grant No. CNS 1149675, ONR Award No. N00014-14-1-0386, and ONR STTR N00014-15-P-1182 and N68335-17-C-0153. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or ONR.

Traditional Satellites

Fault tolerance

- Hardware redundancy
- Rad-hardened processors
- Single-core processors

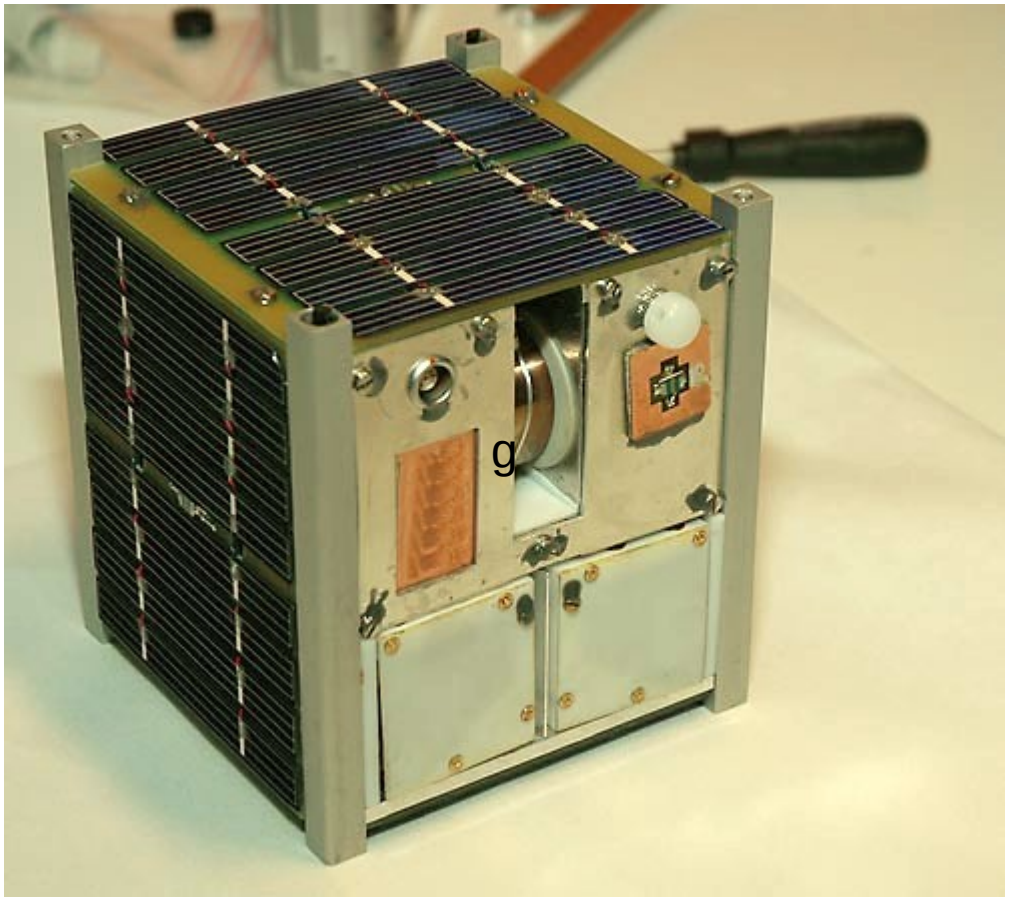
CubeSats



Commodity hardware

- High clock speed
- Multi-core
- Limited hardware reliability features

CubeSats

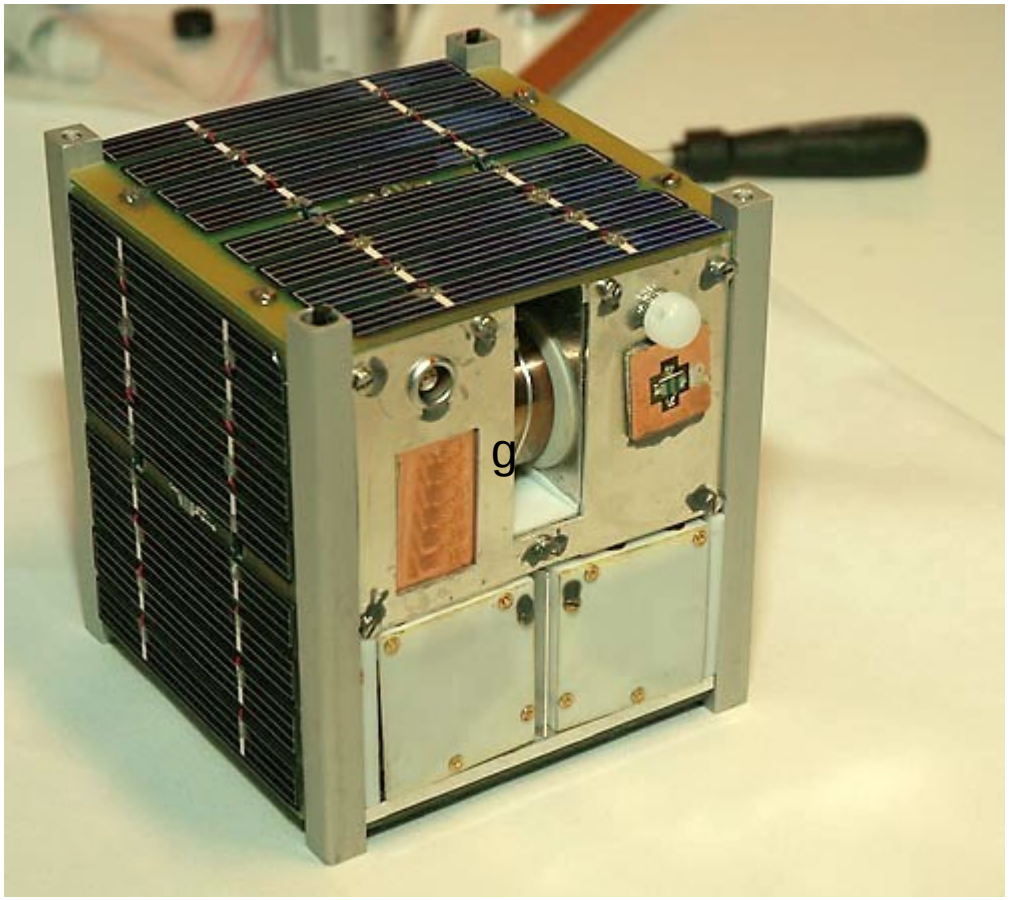


Commodity hardware

- High clock speed
- Multi-core
- Limited hardware reliability features

Spare capacity + no HW reliability → SW reliability

CubeSats



Commodity hardware

- High clock speed
- Multi-core
- Limited hardware reliability features

How to most effectively use the parallelism?

How can we use *extra computational capacity* to increase fault tolerance?

Aspects of SW Fault Tolerance

Detection

Determine *when* system is in an erroneous state

Propagation

How do we contain the scope of the fault

Remediation

How do we return system to a well-defined state

Aspects of SW Fault Tolerance

Detection

Determine *when* system is in an erroneous state

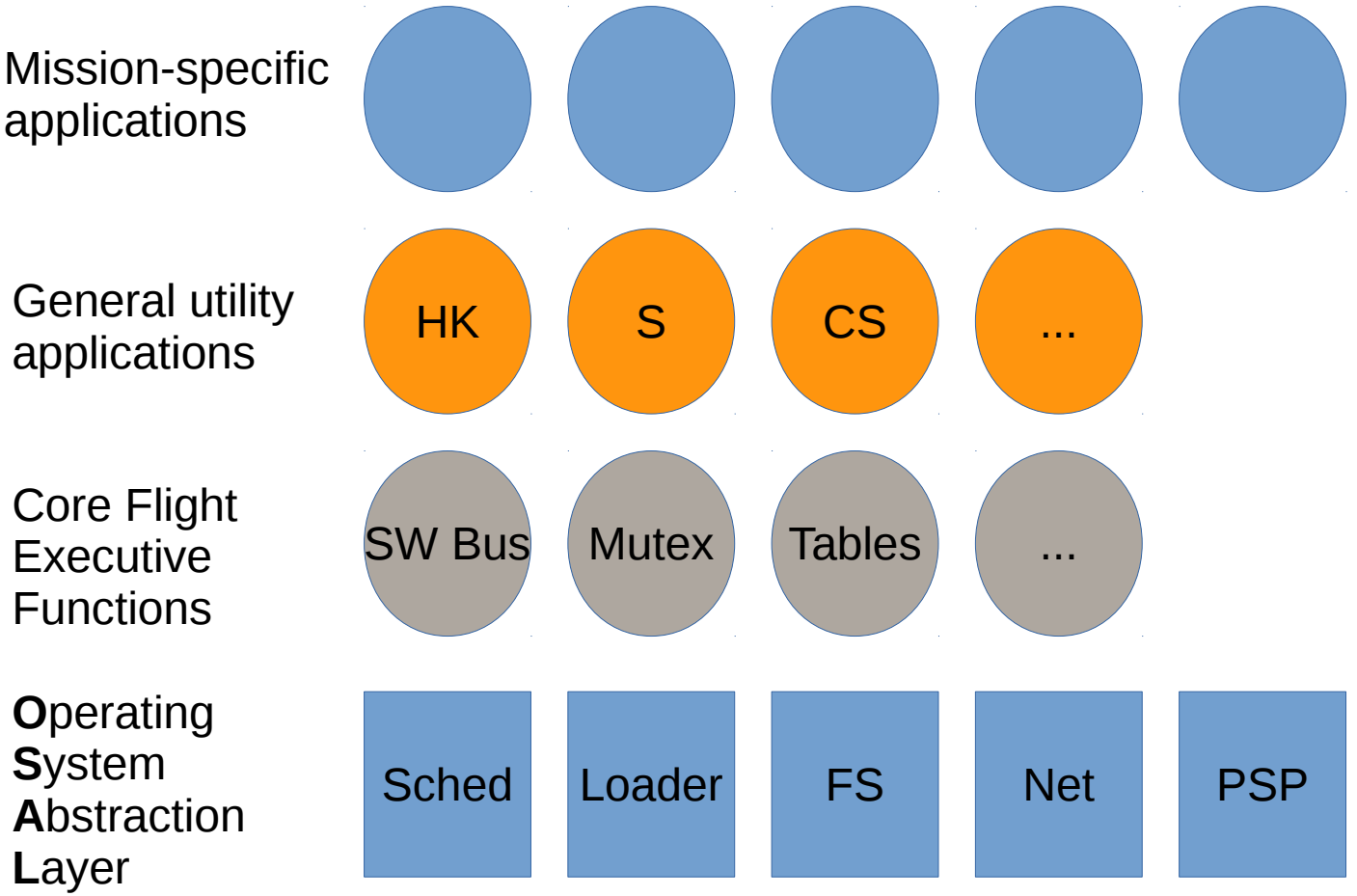
Propagation

How do we contain the scope of the fault

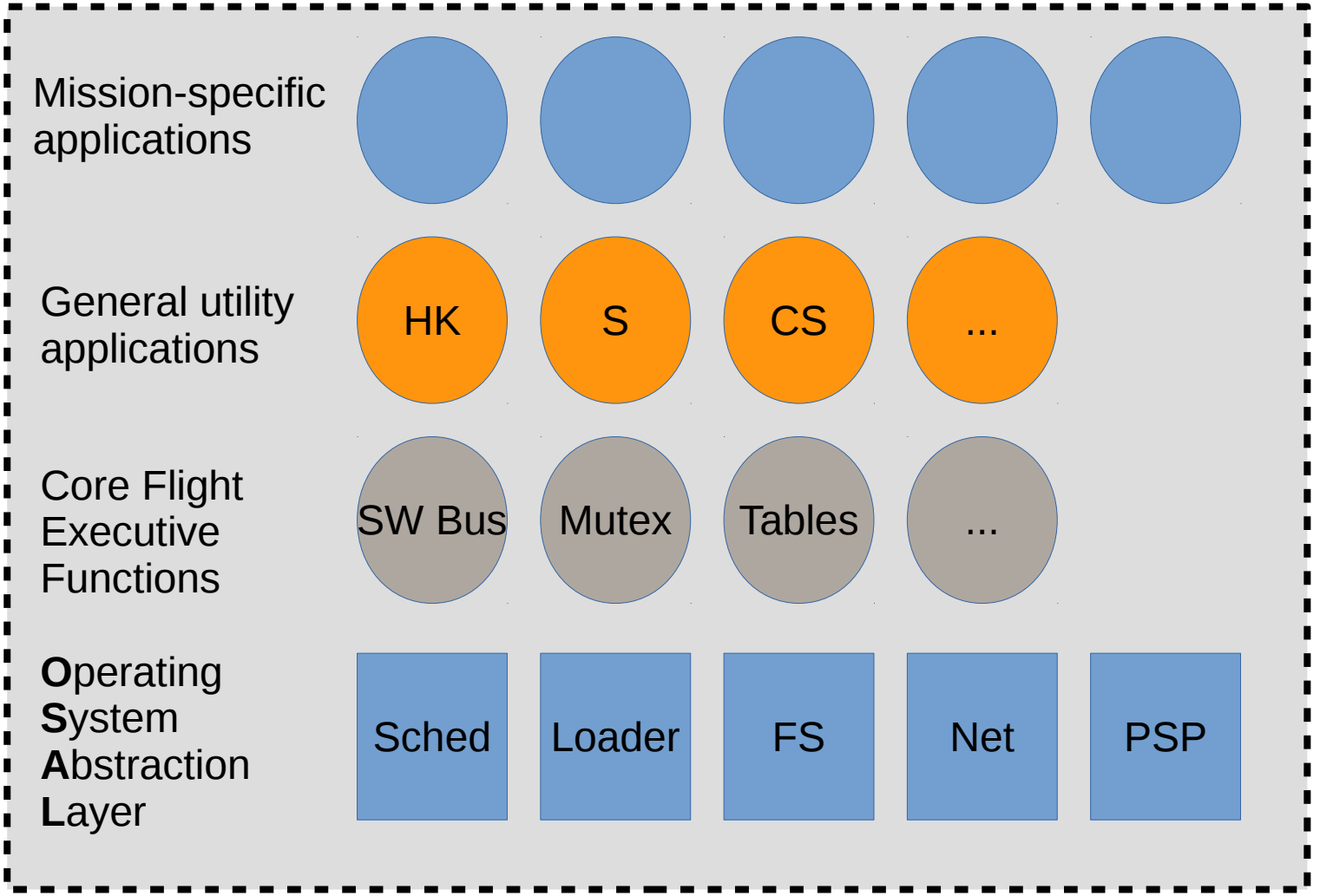
Remediation

How do we return system to a well-defined state

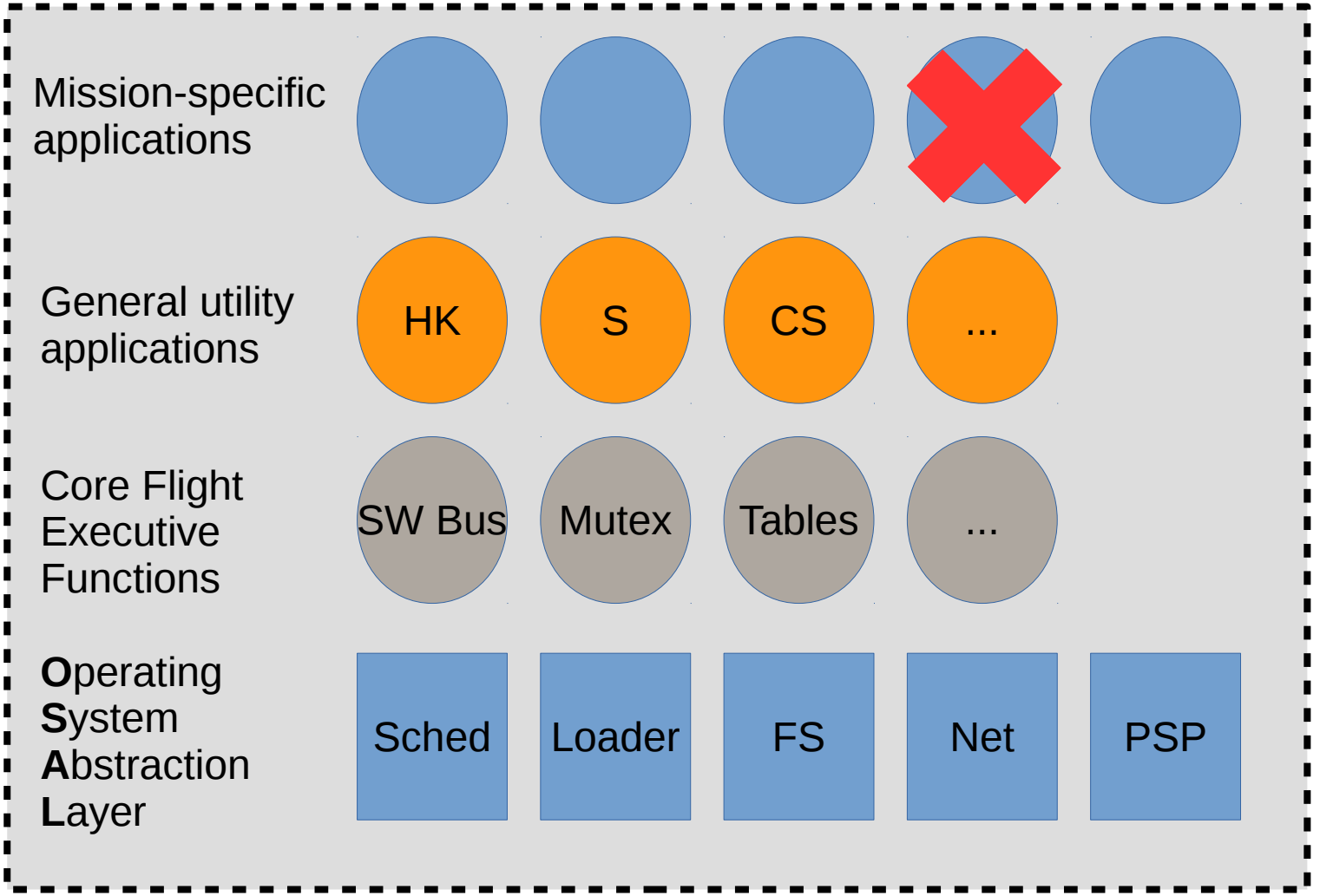
Core Flight System



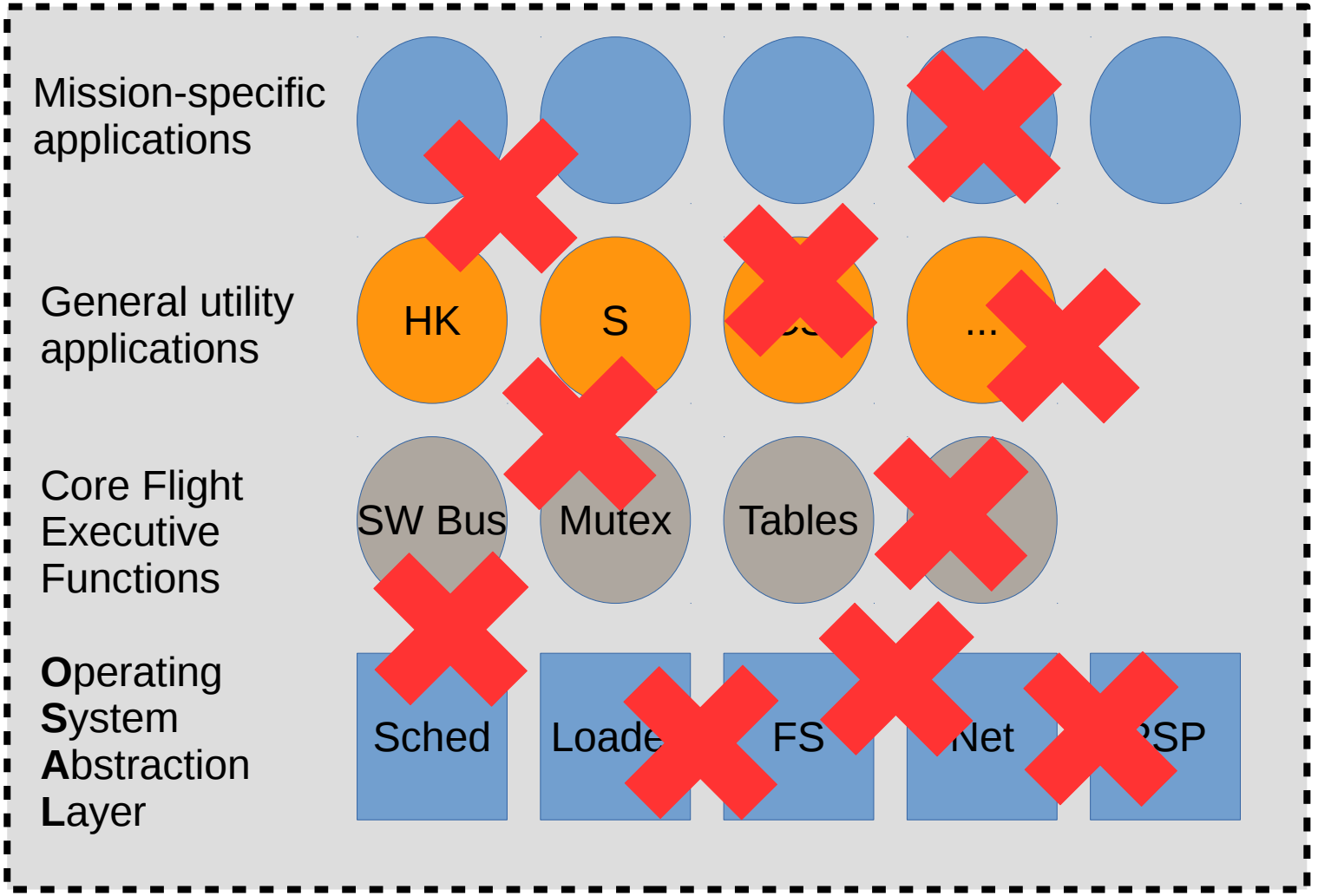
Core Flight System – Faults



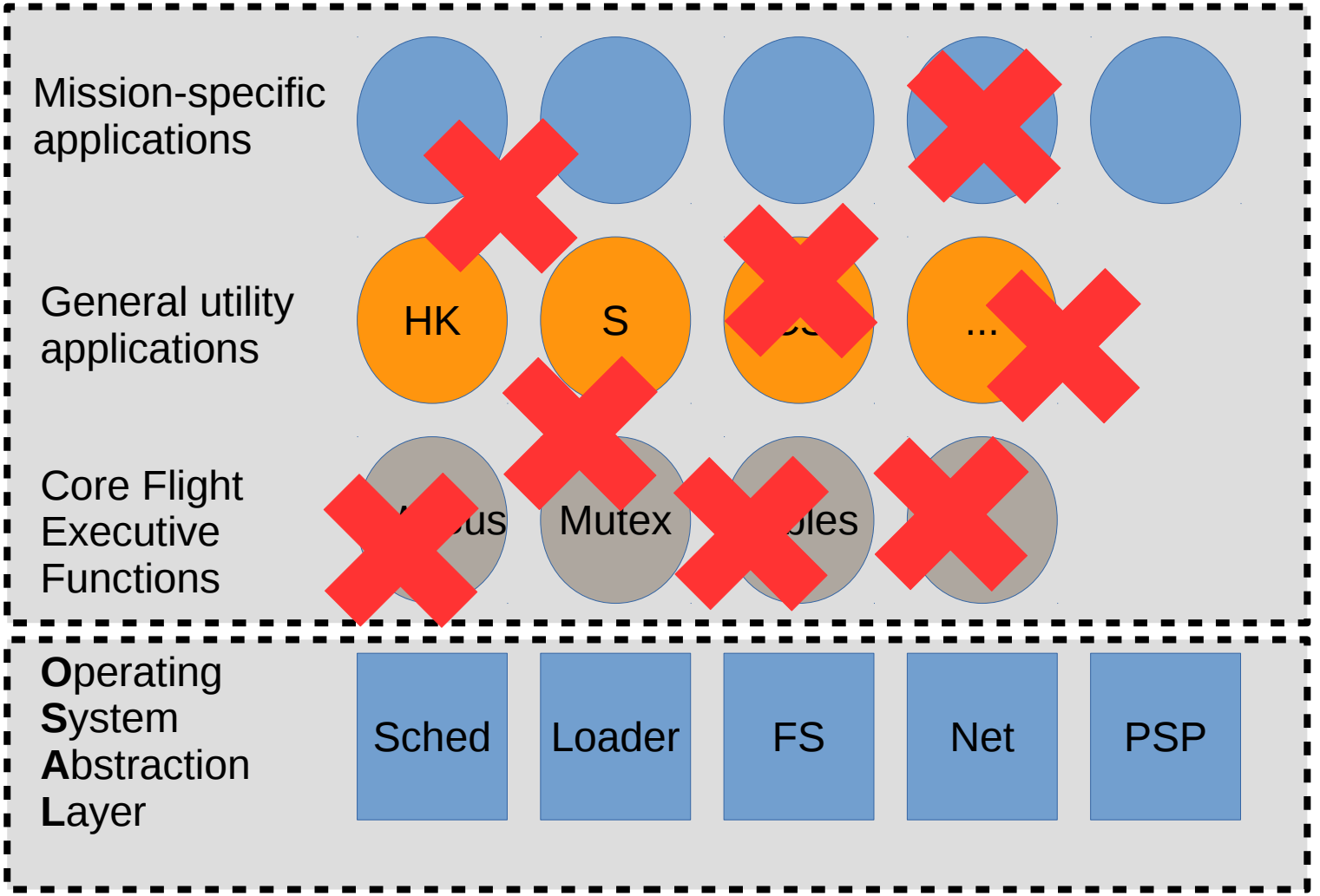
Core Flight System – Faults

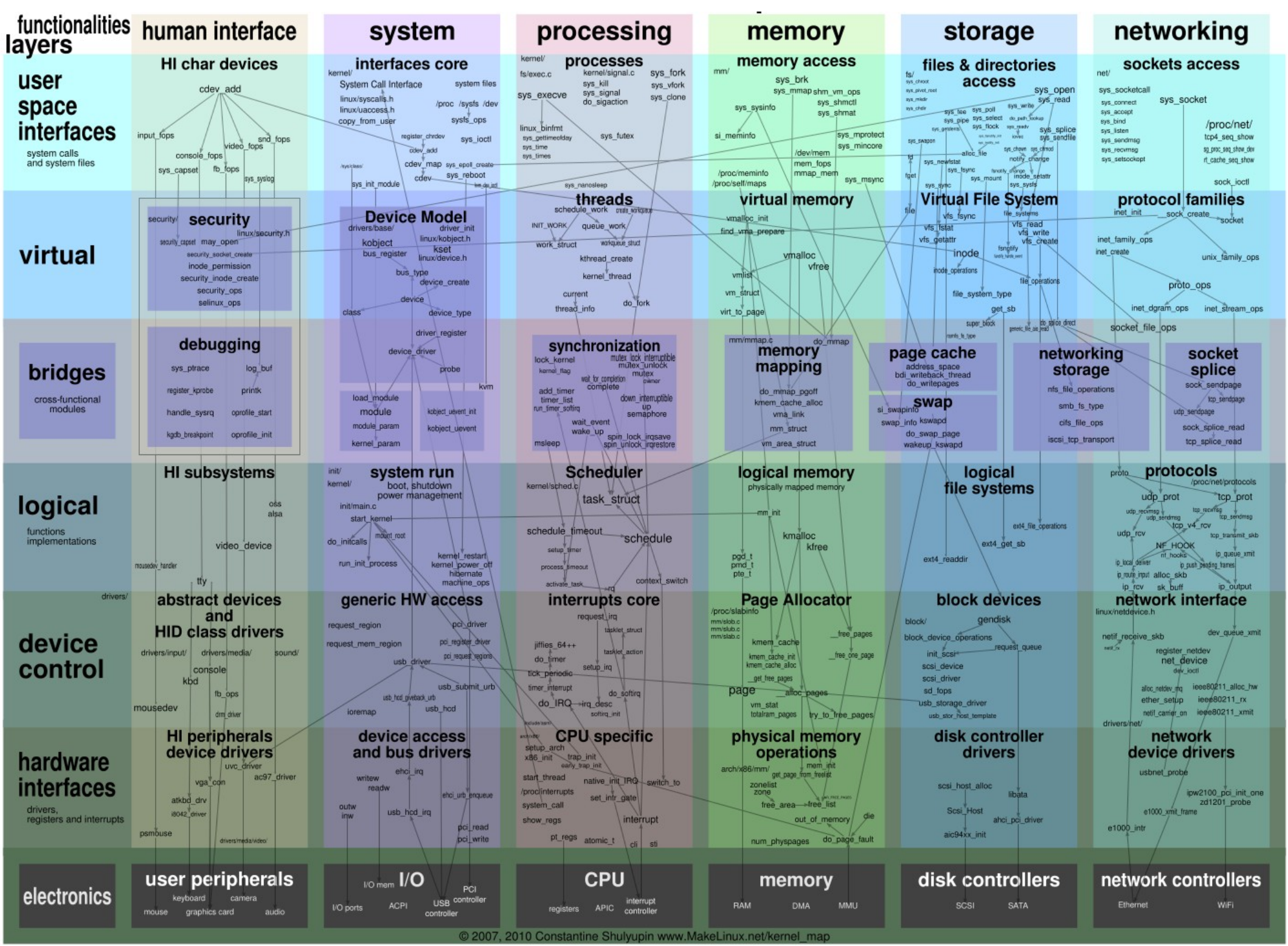


Core Flight System – Faults



Core Flight System – Faults + POSIX





Mission-specific applications

General utility applications

HK

S

CS

...

Core Flight Executive Functions

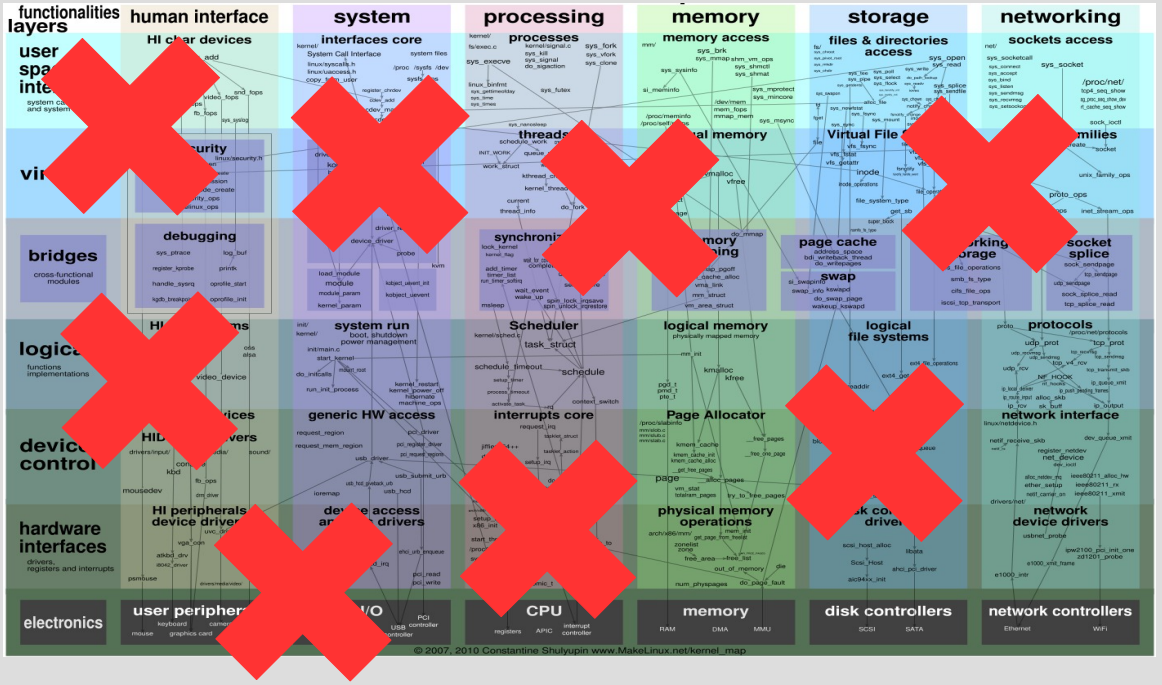
SW Bus

Mutex

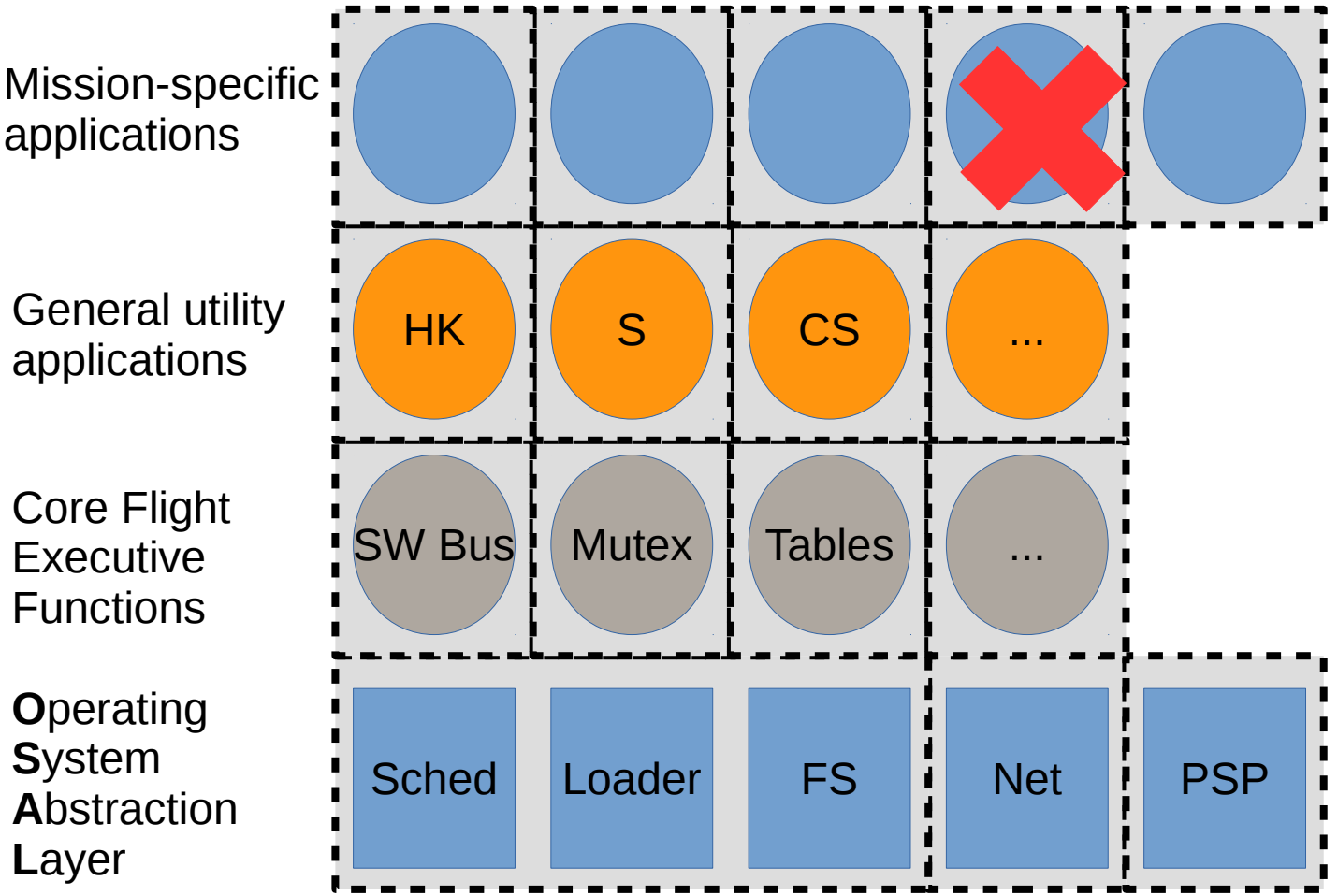
Tables

...

Operating **S**ystem **A**bstraction **L**ayer



Core Flight System



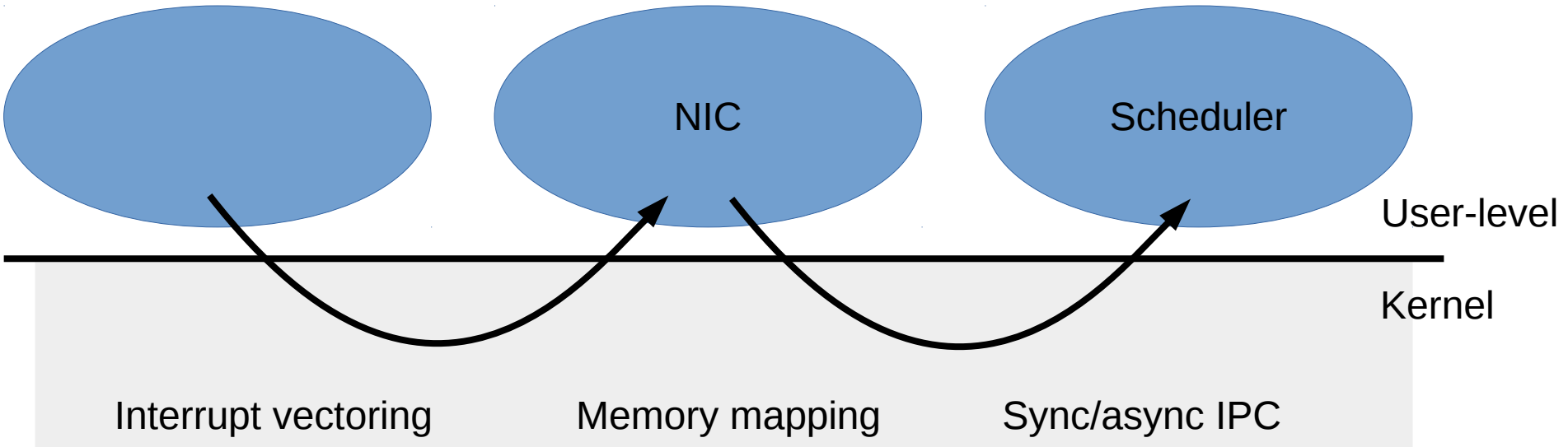
Composite μ-kernel

Small kernel (~7K LoC), real-time focus

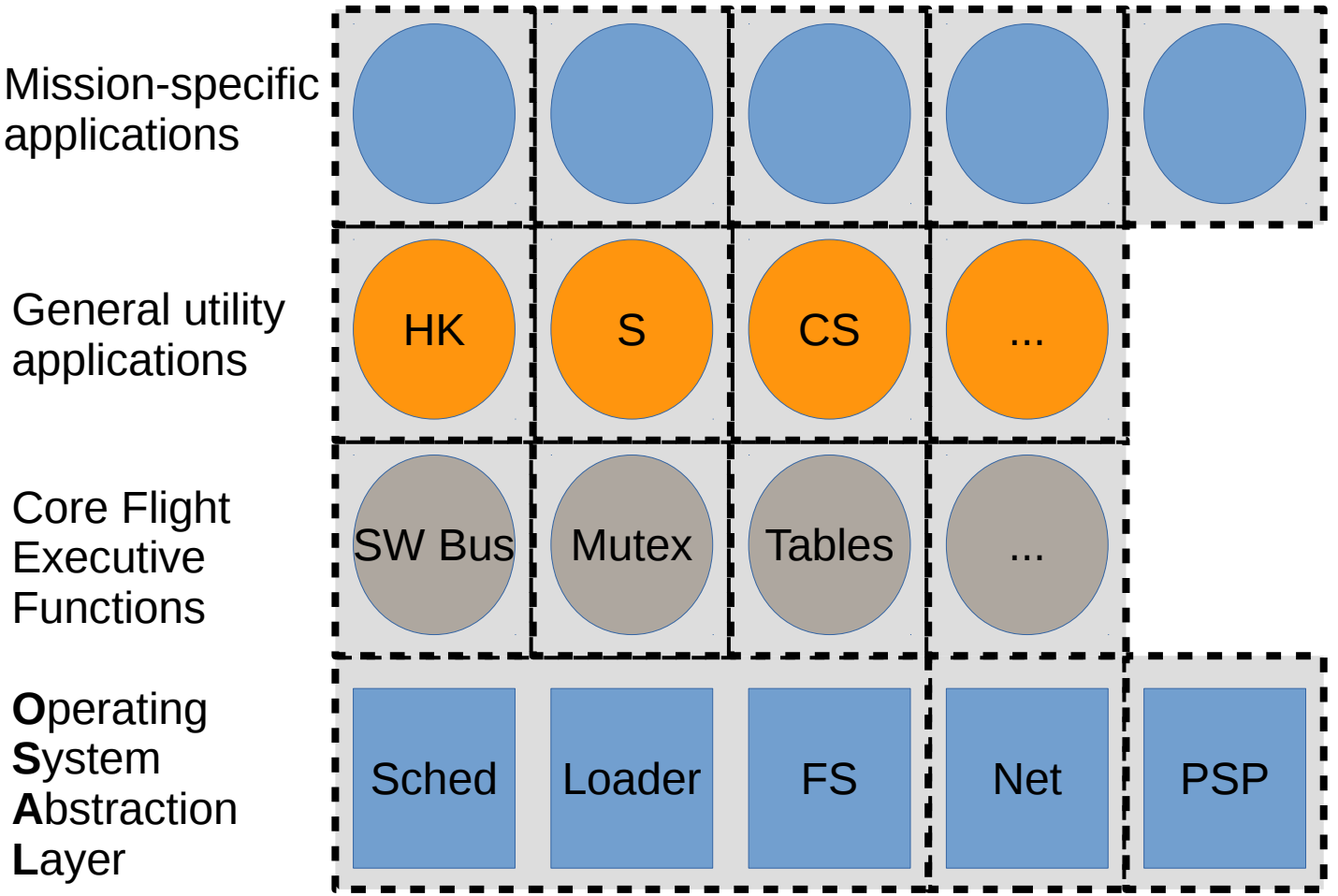
- Focused on IPC between protection domains

Export policies to user-level components

- Scheduling, dev. drivers, memory mgmt, FS, ...



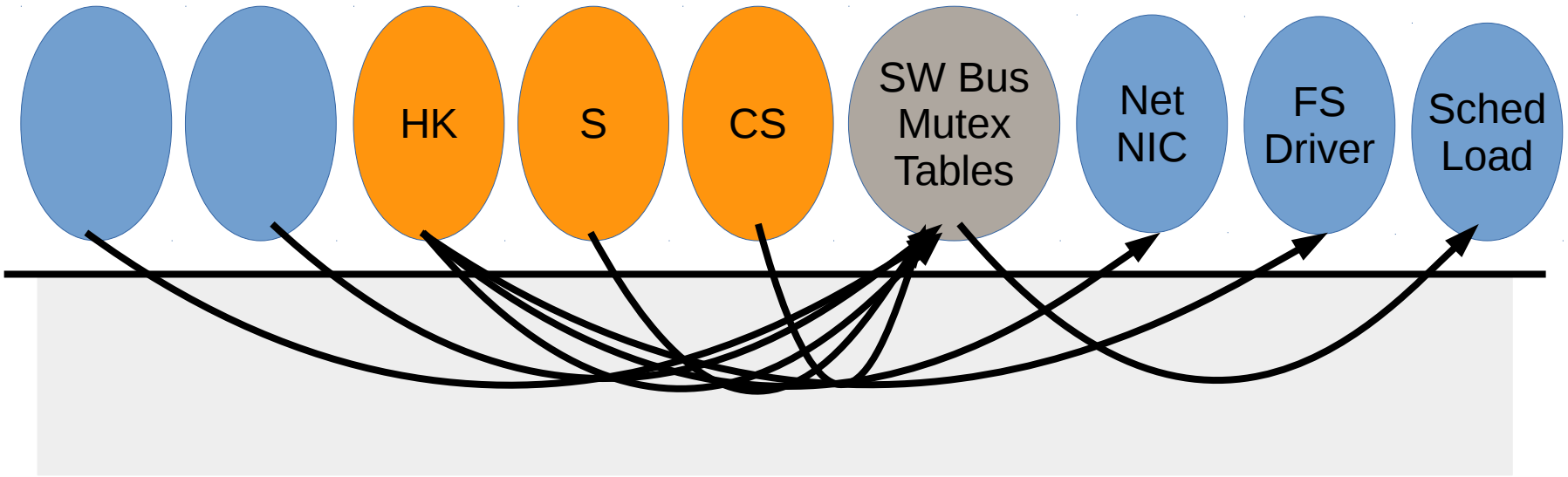
Composite OSAL/PSP



Composite OSAL/PSP

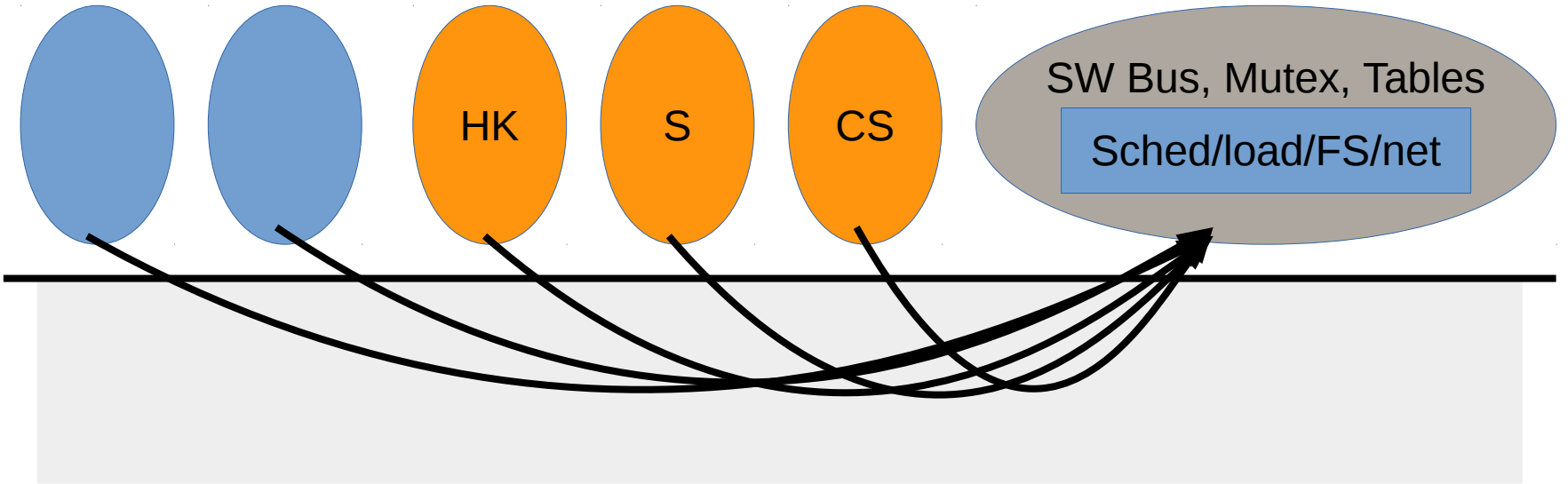
- Communication explicitly controlled by design
- IPC and scheduling are fast:

	Composite	Linux
2-way IPC	**700** cycles	**600** (syscall), **3500** (pipes)
Thd Dispatch	**300** cycles	**1800** (yield)



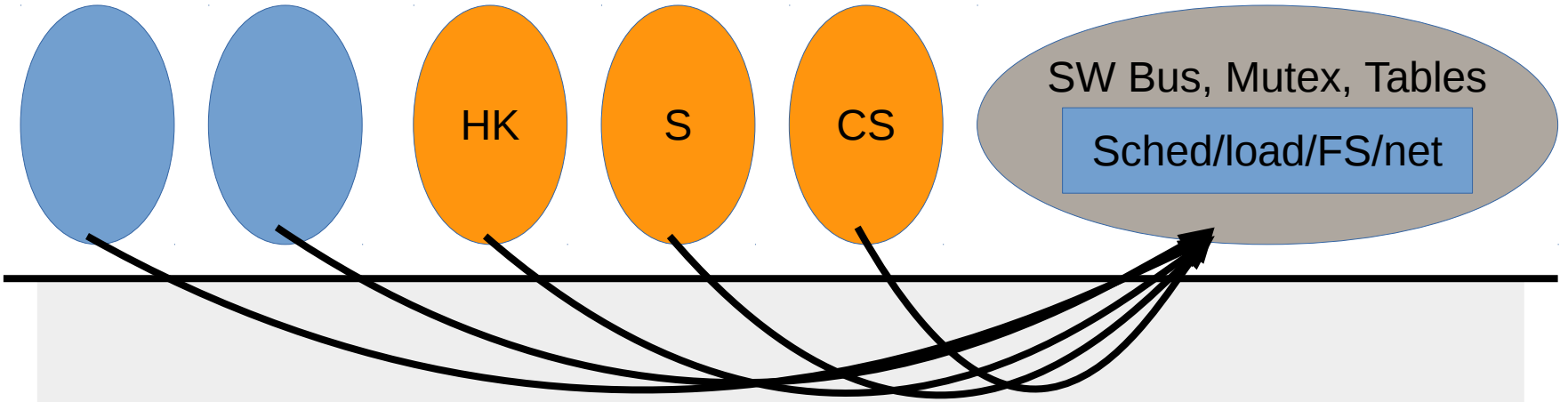
Composite OSAL/PSP – *Current*

- Fixed priority preemptive scheduling
- RAM-based FS
- Application loader:
  - Into *shared* protection domain
  - Into *separate* protection domains



Composite OSAL/PSP – *Current*

- Lines of C Code: < **4000** LoC
- OSAL unit tests: > **89**% successful
  - oscore/osfile/osfilesys/osloader, 15% not relevant (OS call failure)
- In progress:
  - serialization/deserialization of OSAL arguments
  - increasing application support



Aspects of SW Fault Tolerance

Detection

Determine *when* system is in an erroneous state

Propagation

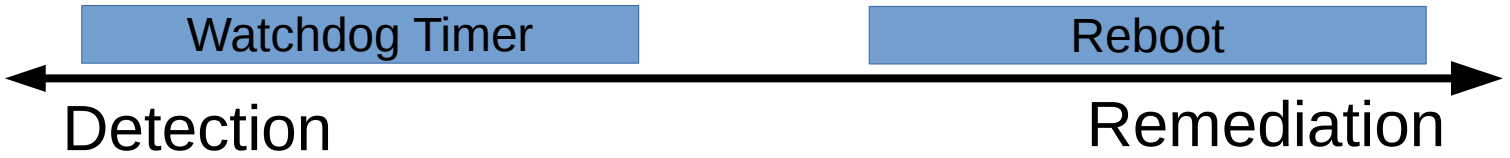
How do we contain the scope of the fault

Remediation

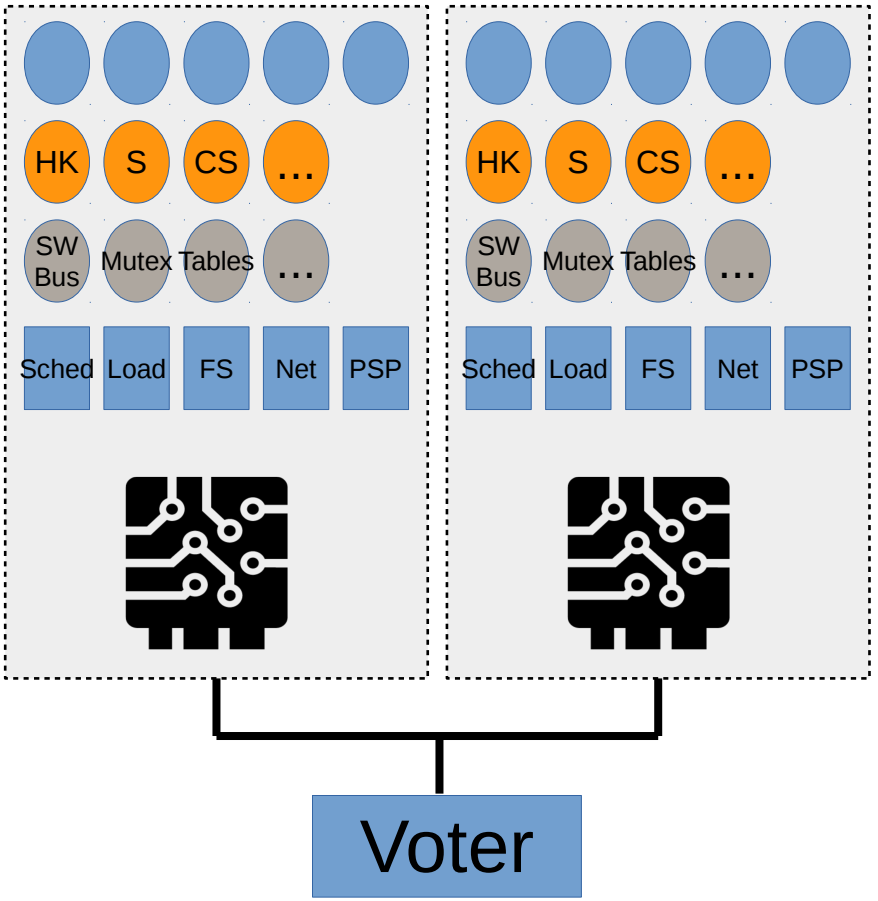
How do we return system to a well-defined state

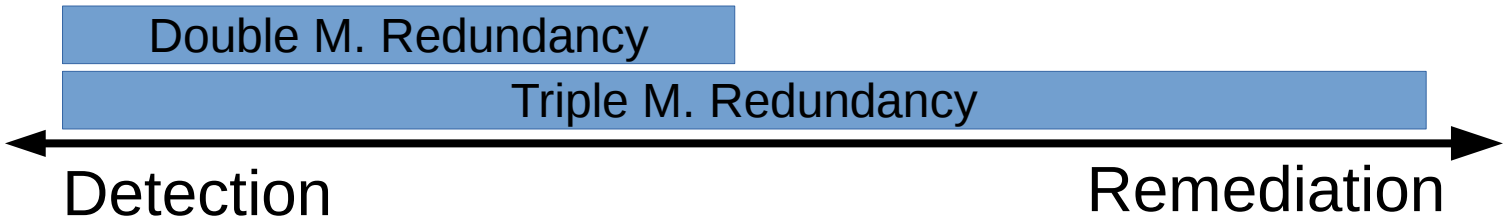
Watchdog Timer

- Applications
  - periodically declare successful execution
- Every watchdog timer (1-10 seconds):
  - Have all applications and system components checked in?
  - No: reboot!

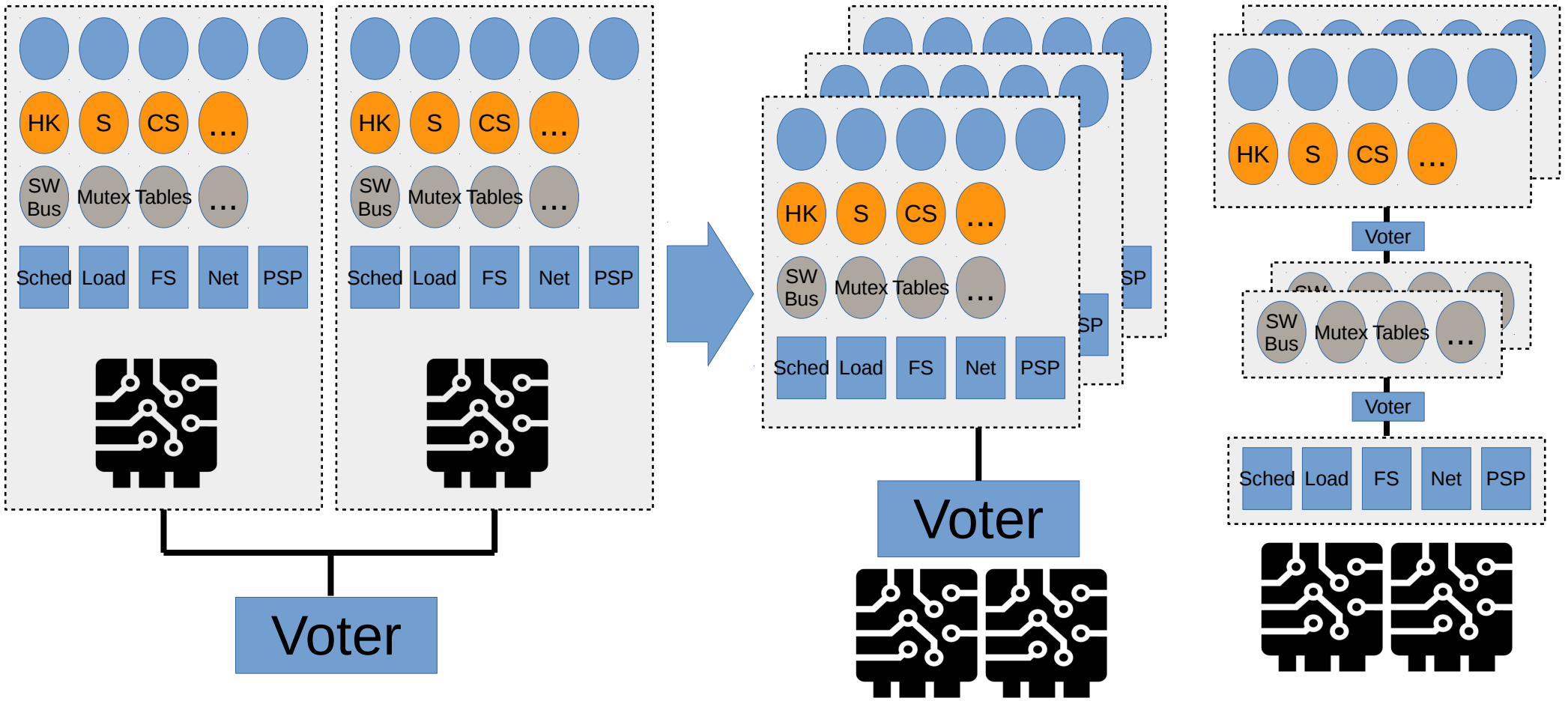


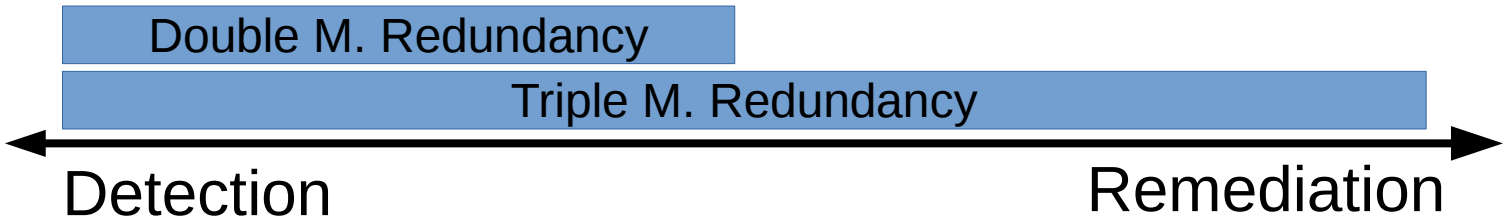
Redundant Execution



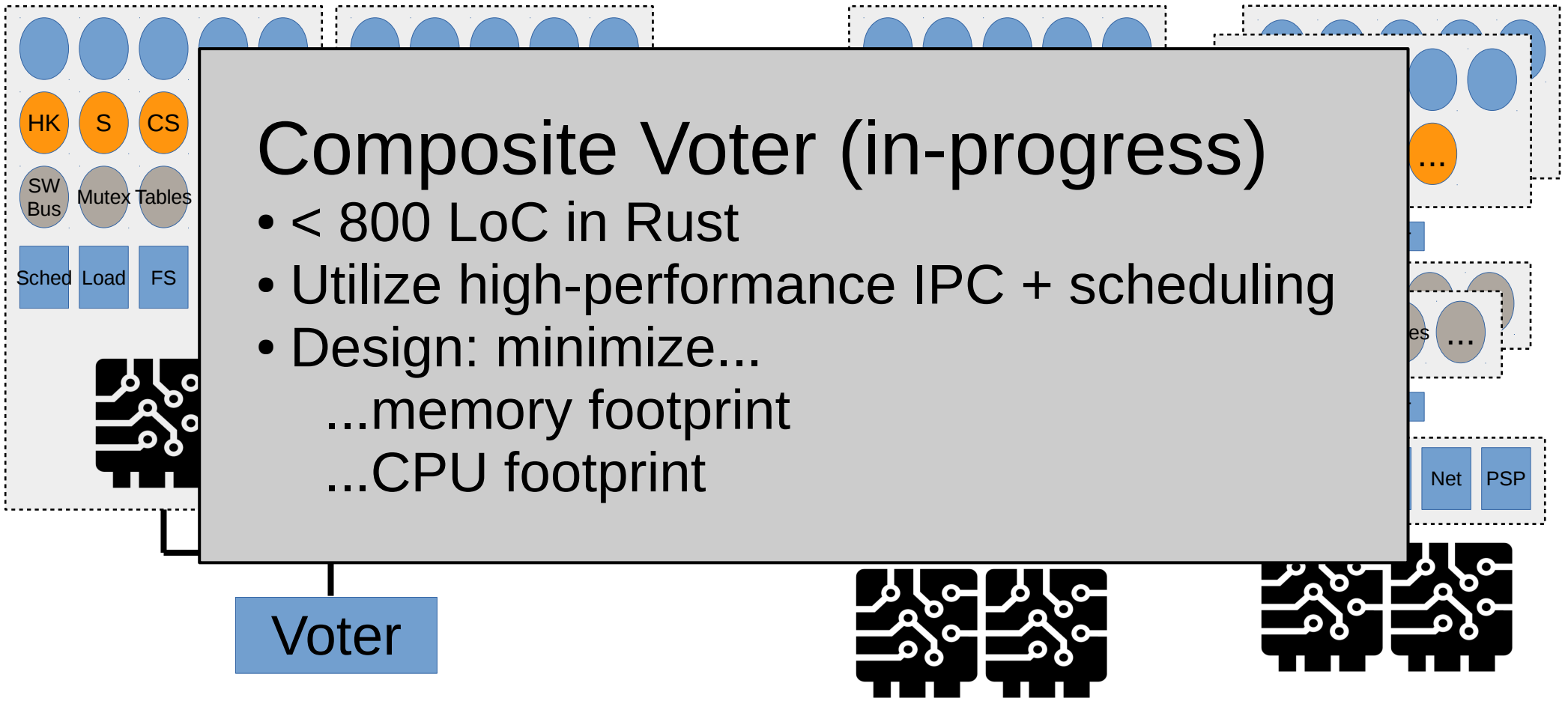


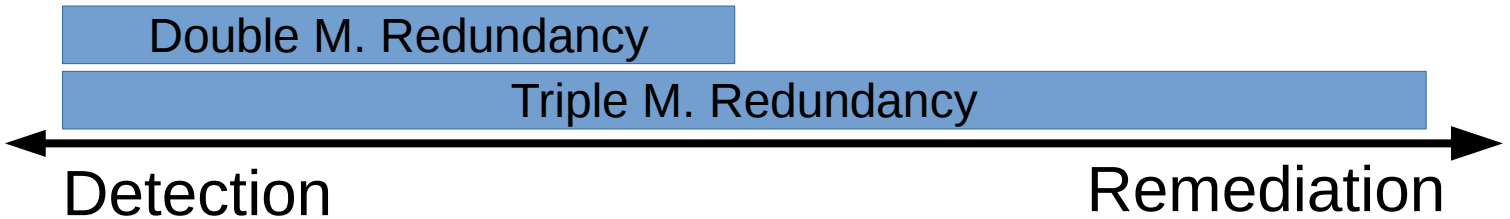
Redundant Execution



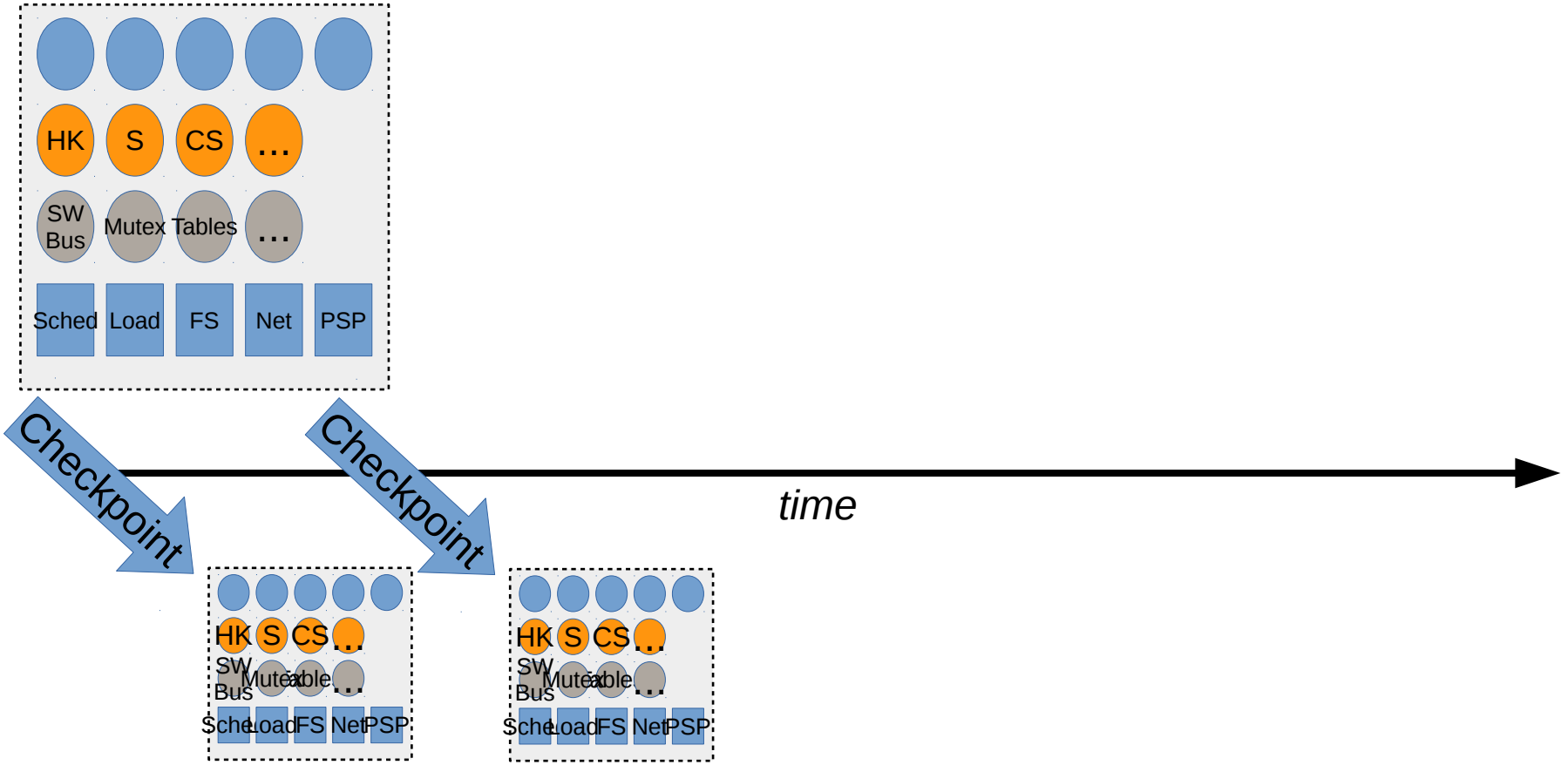


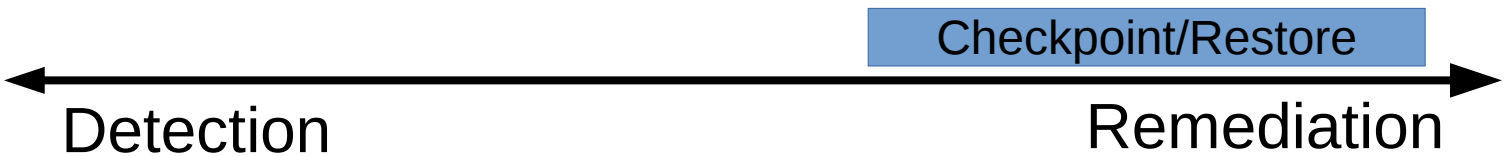
Redundant Execution



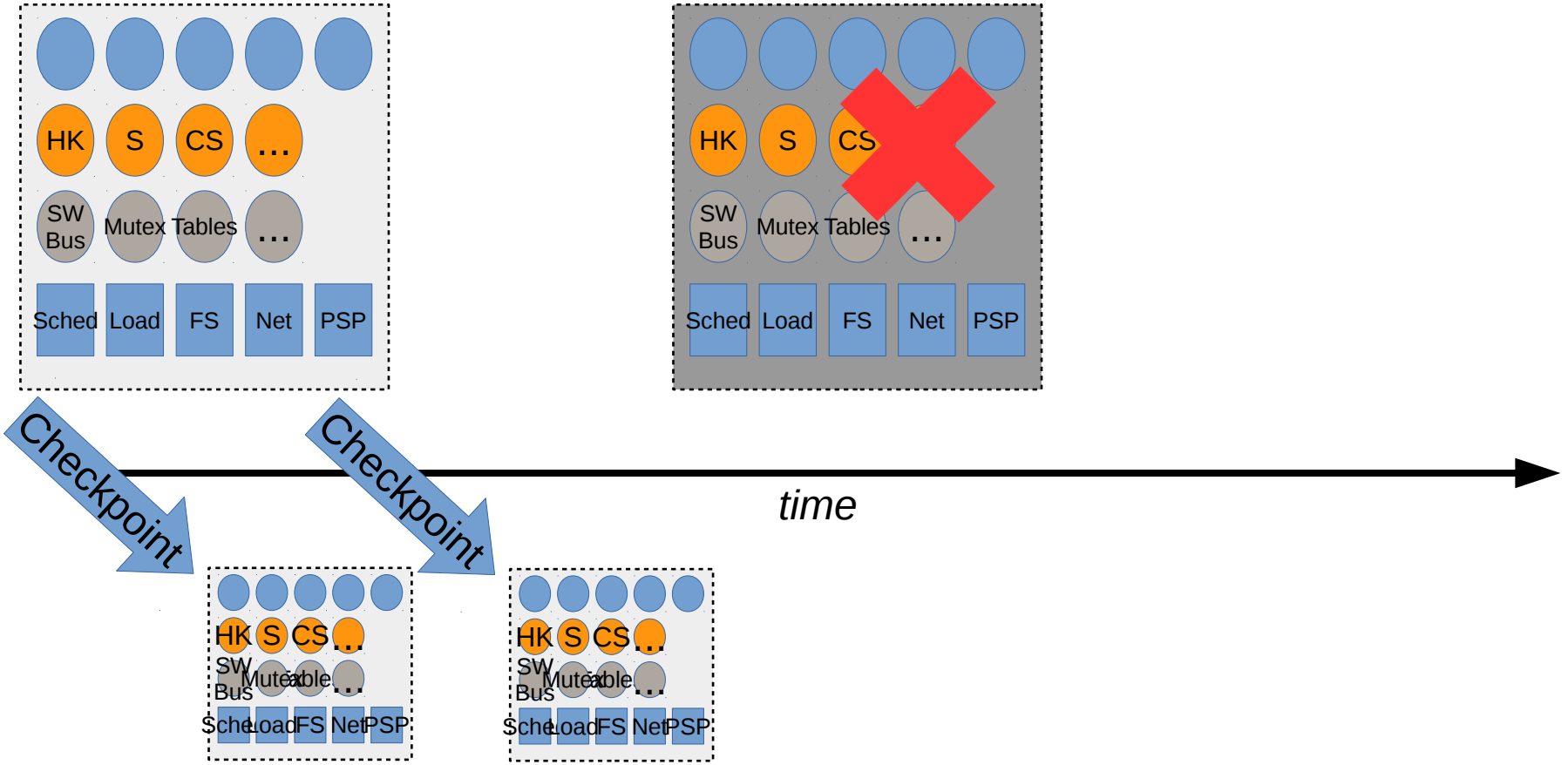


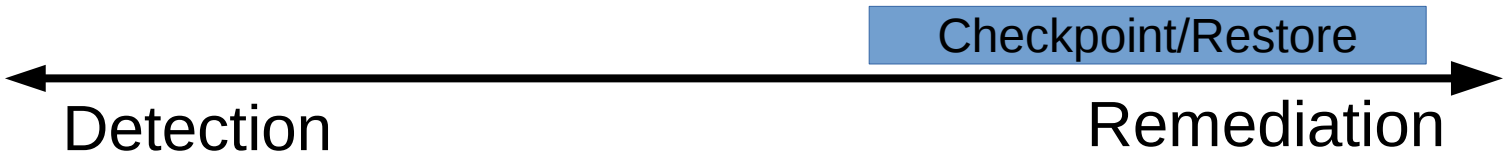
Checkpoint/Restore



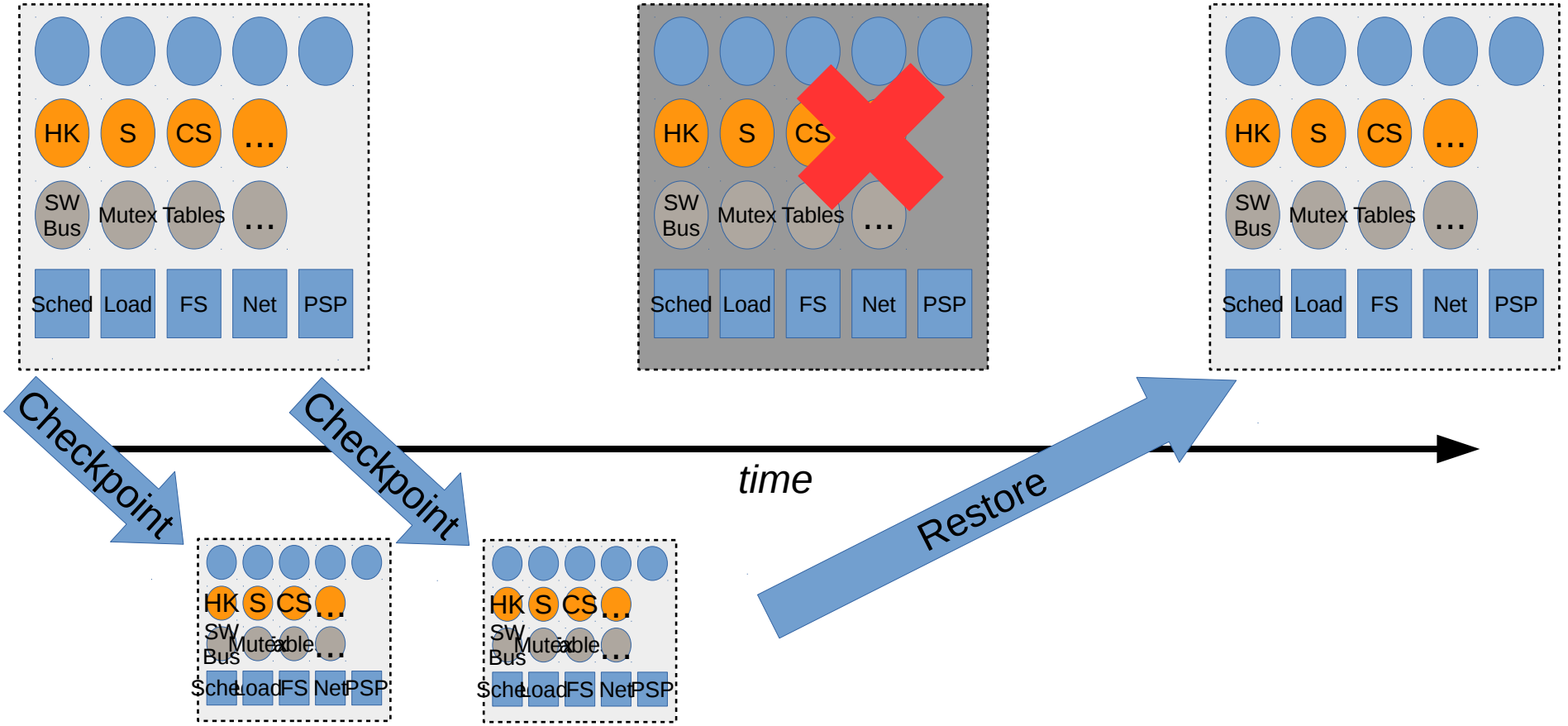


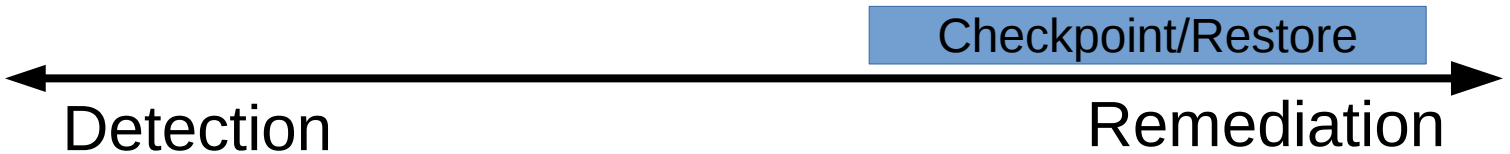
Checkpoint/Restore



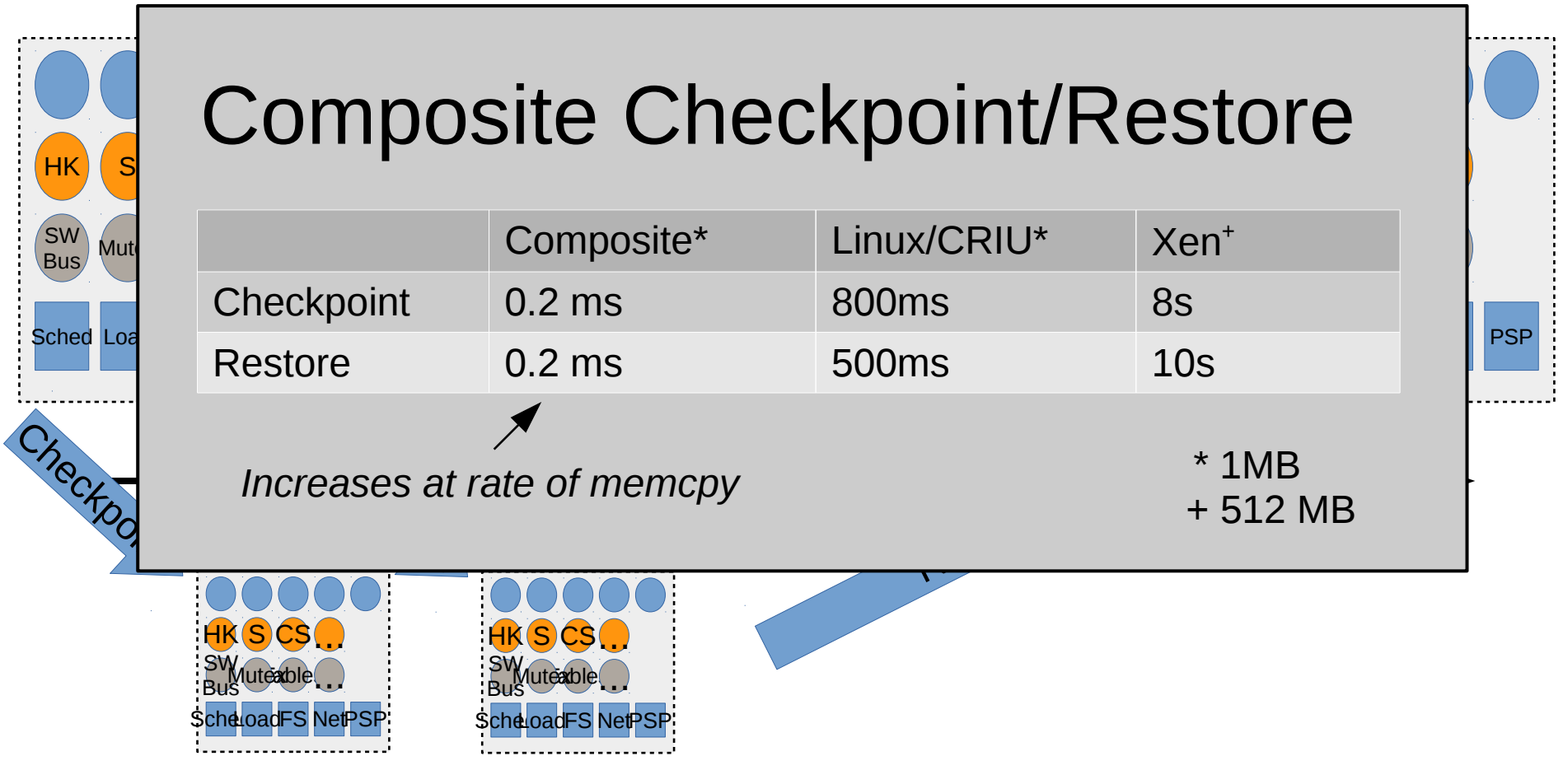


Checkpoint/Restore





Checkpoint/Restore



Composite Checkpoint/Restore

	Composite*	Linux/CRIU*	Xen+
Checkpoint	0.2 ms	800ms	8s
Restore	0.2 ms	500ms	10s

Increases at rate of memcpy

* 1MB
\+ 512 MB

Checkpoint/Restore

Detection ⟷ Remediation

Computational Crash Cart

Recover system-level components upon failure

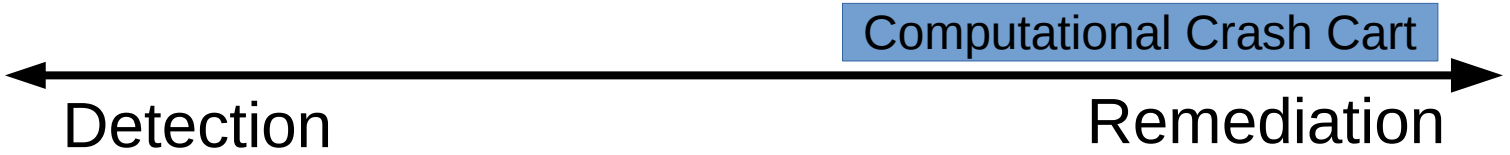
- Record summary of component comms
- Reboot component + re-establish state

Focus on *real-time*

- 10s of **micro**-second recovery time

Complementary to application-level reliability

- Checkpoint/Redundant execution



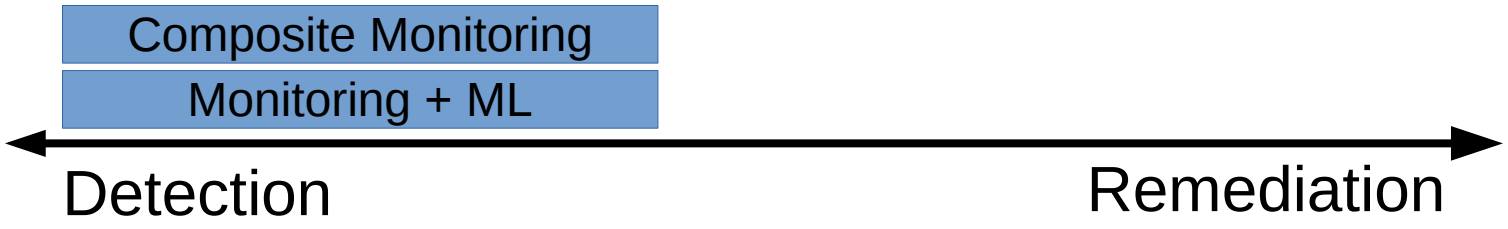
Monitoring for Detection

Monitor/log system interactions and timing

- API calls, context switches, interrupts, …

Process log

- *Interactions deviate from system model?*
- *Interactions statistically deviate from historically correct behaviors?*

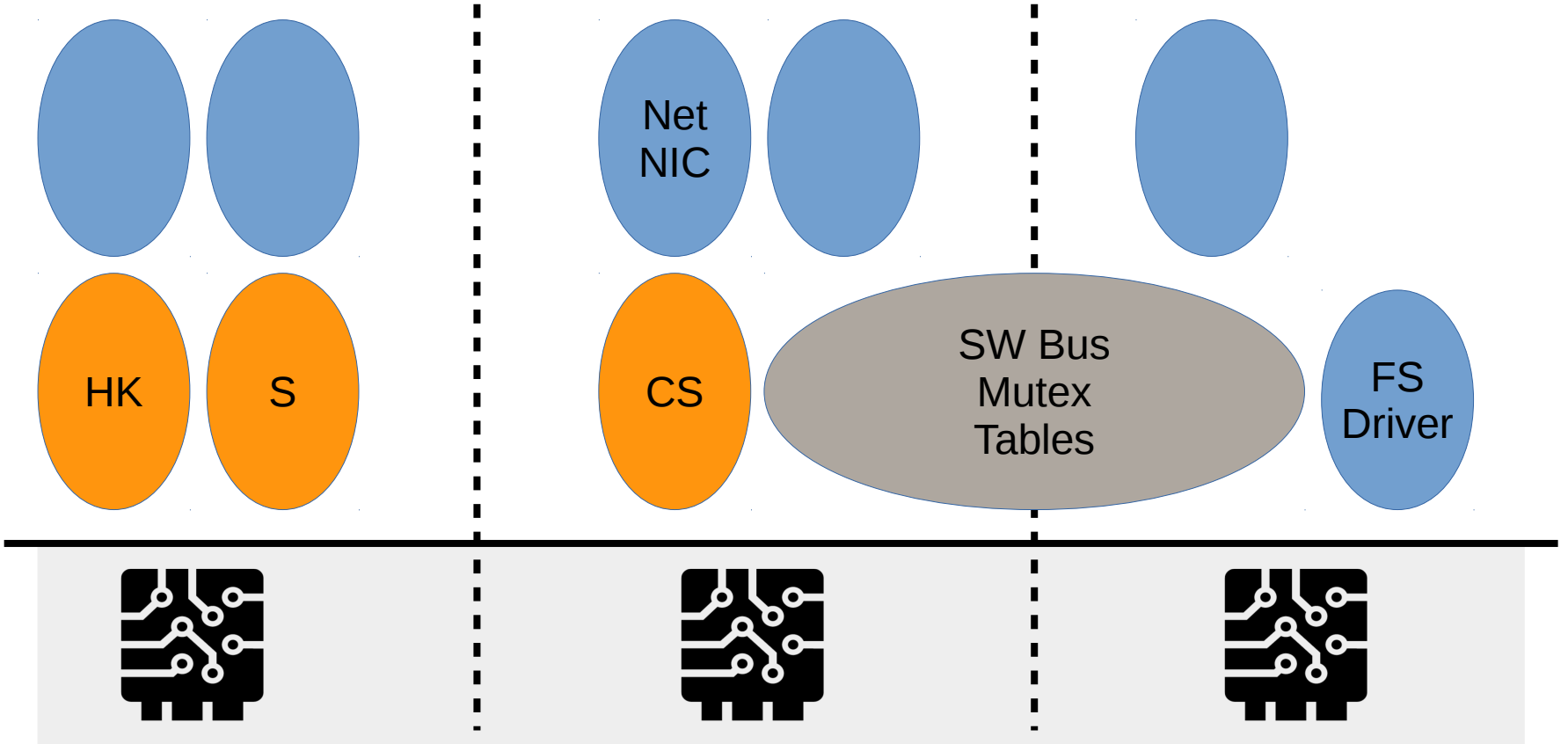


How can we effectively use the parallelism of commodity CPUs?

Composite + Parallelism

Kernel designed to be *lock-less*

- Kernel operations are all wait-free → real-time
- IPC core-local, or inter-core



Composite + Parallelism

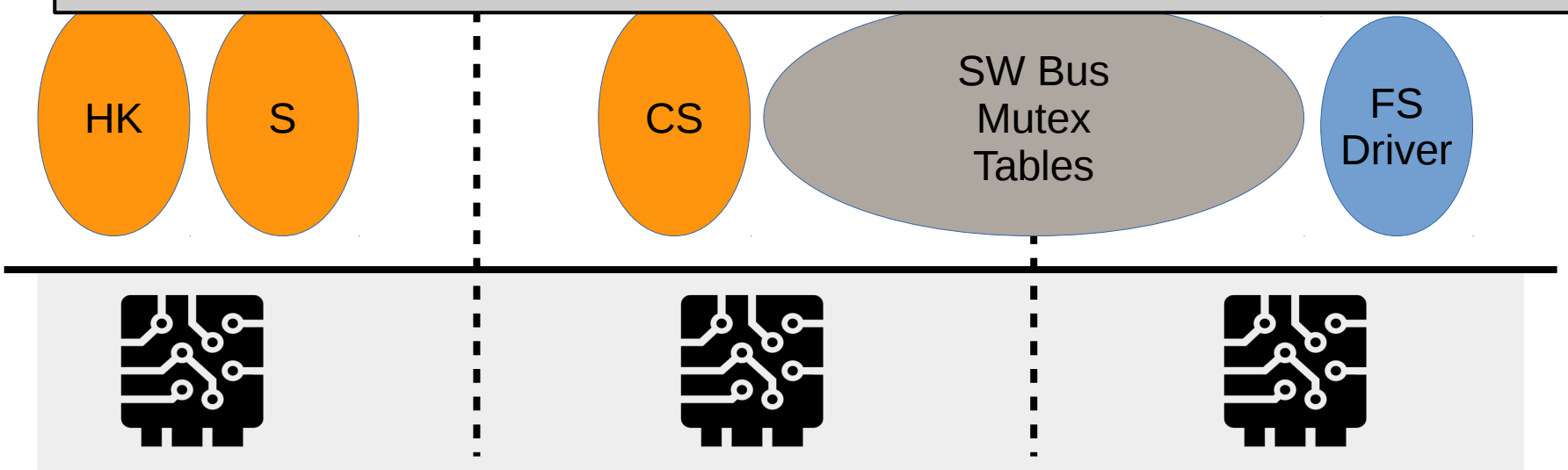
Ke

- [illegible]
- [illegible]

Composite parallelism orchestration

Example: OpenMP fork/join parallelism

- 2-40x decrease in worst-case inter-core communication latencies
- Up to 40 cores, across 4 sockets



CubeSats: Fresh View on cFE

How can we effectively

- utilize new resource availability, and
- provide SW fault tolerance?

→ **Composite OSAL** enables new options

Where do we go from here?

- Need your feedback...

? || /* */


GW

- Computational Crash Cart, Checkpointing:
  http://www2.seas.gwu.edu/~gparmer/publications/rtss13_c3.pdf
- Model-based event monitoring:
  http://www2.seas.gwu.edu/~gparmer/publications/rtas15cmon_extended.pdf
- ML-based event monitoring:
  http://www2.seas.gwu.edu/~gparmer/publications/certs16caml.pdf
- Micro-benchmarks and virtualization:
  http://www2.seas.gwu.edu/~gparmer/publications/rtss17tcaps.pdf
- Lock-free, predictable kernel:
  http://www2.seas.gwu.edu/~gparmer/publications/rtas15speck.pdf
- OpenMP Fork/Join parallelism:
  http://www2.seas.gwu.edu/~gparmer/publications/rtas14_fjos.pdf