The State of Composite: a Customizable Component-Based OS for Predictable, Reliable, and Scalable Computation

Gabriel Parmer

gparmer@gwu.edu
The George Washington University (GWU)

OSPERT 2013

Researchers include
Qi Wang, Jiguo Song, Jakob Kaivo,
Andrew Sweeney, John Wittrock, ...

Embedded Systems

Past:

- single, simple task
- uni-processor
- fault-tolerance ignored (reboot), or custom

Present/Future:

- consolidation
- certification
- multi-/many-core
- increased faults due to shrinking manufacturing processes

Embedded OSes

Past:

- single memory protection domain
- threads, FP scheduling, semaphores, mailboxes, timing
- FreeRTOS, OSEK, ...

Challenges of the Present/Future:

- spatial + temporal isolation
- system composition from independently certifiable pieces
- intra- and inter-task parallelism
- reliability built-in

Embedded OSes

Past:

- single memory protection domain
- threads, FP scheduling, semaphores, mailboxes, timing
- FreeRTOS, OSEK, ...

Challenges of the Present/Future:

- spatial + temporal isolation
- system composition from independently certifiable pieces
- intra- and inter-task parallelism
- reliability built-in

Challenge: predictability
Challenge: maintaining system simplicity

Linux kernel map

2.6.36

functionalities
layers

	human interface	system	processing	memory	storage	networking
user space interfaces system calls and system files	HI char devices	interfaces core	processes	memory access	files & directories access	sockets access
virtual	security debugging	Device Model	threads	virtual memory	Virtual File System	protocol families
bridges cross-functional modules			synchronization	memory mapping	page cache swap	networking storage socket splice
logical functions implementations	HI subsystems	system run	Scheduler	logical memory	logical file systems	protocols
device control	abstract devices and HID class drivers	generic HW access	interrupts core	Page Allocator	block devices	network interface
hardware interfaces drivers, registers and interrupts	HI peripherals device drivers	device access and bus drivers	CPU specific	physical memory operations	disk controller drivers	network device drivers
electronics	user peripherals keyboard, camera, mouse, graphics card, audio	I/O I/O mem, I/O ports, ACPI, USB controller, PCI controller	CPU registers, APIC, interrupt controller	memory RAM, DMA, MMU	disk controllers SCSI, SATA	network controllers Ethernet, WiFi

Map of GNU/Linux OS and FOSS

layers \ functions	user	system	data	net
presentation high level and general purpose user programs /usr/bin /usr/lib	**desktops** xfce Gnome KDE **office** OpenOffice KOffice Evolution LaTeX	**packaging** kpackage synaptic yum portage apt rpm urpmi dpkg	**file management** Thunar Konqueror Nautilus Krusader K3b gnome-commander tar bzip2 Ark gzip	**net clients** Firefox KMail Thunderbird Pidgin Kopete Mutt mail
application application specific programs /usr/bin /usr/lib	**audio, video, graphics** MPlayer Amarok FFmpeg GStreamer VLC Xine GIMP krita Inkscape Blender	**development** Emacs Anjuta VIM KDevelop Eclipse bugzilla subversion gdb make gcc binutils	**text processing** diff Meld kdiff3 grep sed nano kate gedit textutils: uniq sort comm join cat paste	**net utilities** Wireshark tcpdump wget netcat curl traceroute ping
engines services servers interpreters infrastructure /usr/sbin /usr/lib	**GUI** metacity gdm XX.org kdm GTK+ Qt lpd cups	**system services** klogd acpid syslogd crond D-Bus udev init hotplug hald	**interpreters** Perl PHP awk Python **DBMS** PostgreSQL SQLite MySQL	**data & net** LAMP aMule rsync FTP NFS samba iSCSI **net servers** sshd Apache postfix inetd portmap named NetworkManager
control administration and basic access /sbin /bin	**user access** su man chown adduser bash chmod echo coreutils: pwd printf	**system adm** lsusb lspci lshal top ps jobs nice chkconfig kill printenv	**memory** procps free slabtop vmstat **storage config** stat file sync lvm2 findutils ls mkdir mkfs fdisk mount ln dd df du cp rm	**network adm** iwconfig ip iptables netstat route ifconfig host socklist
foundation base libraries, kernel and resources /etc /boot /sbin /lib	libselinux login libcrypt getty console HID	ld.so librt pthread libdl processes initrd /lib/modules	libstdc++ libxml2 libexpat zlib **GNU C Lib** libm files **Linux kernel**	libssl libresolv sockets protocols
hardware	**user peripherals**	**PCI USB**	**RAM**	**storage**	**Ethernet WiFi**

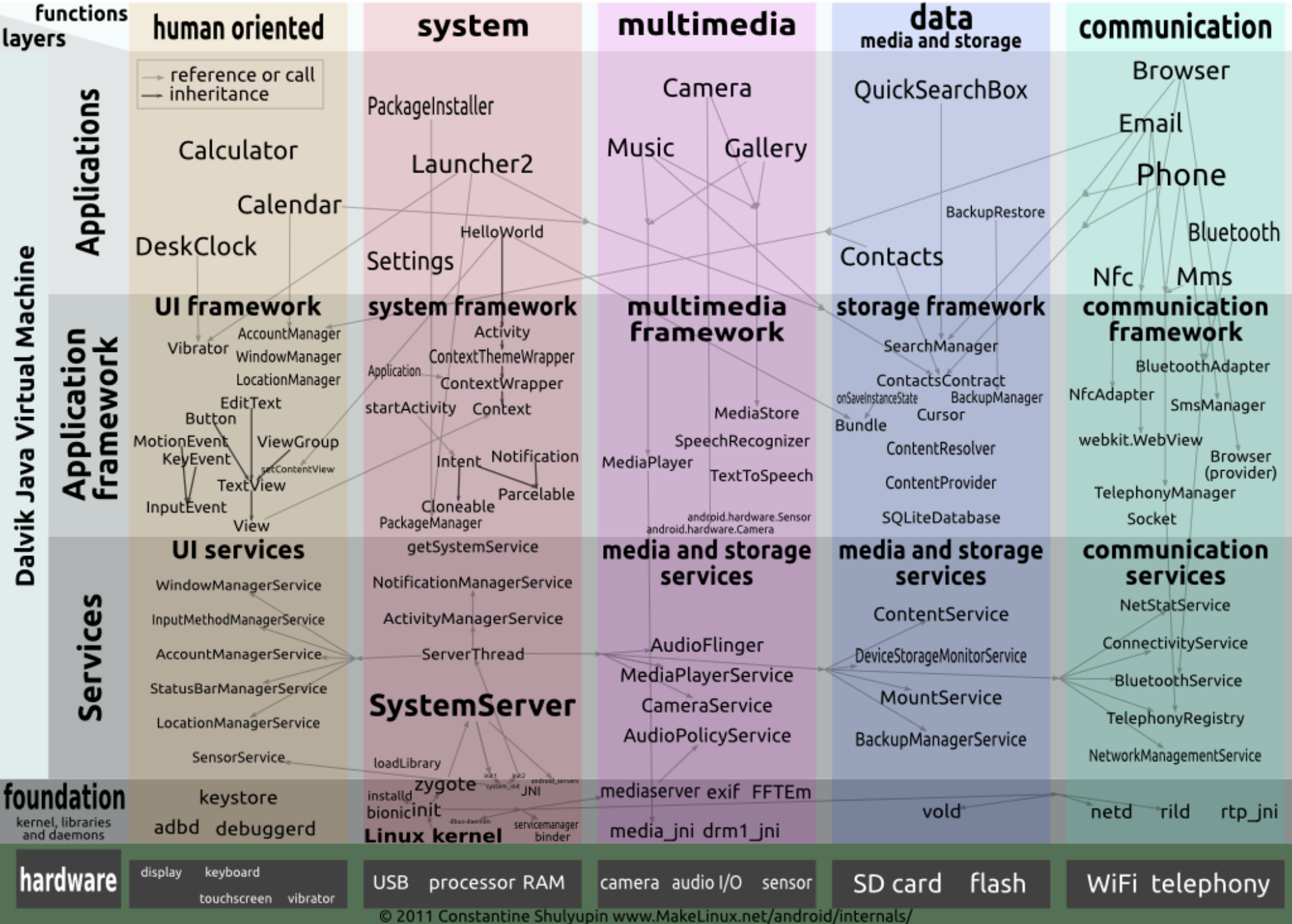
boot

power

GRUB

Android Internals

API Level 9



The COMPOSITE Component-Based OS

System policies/abstractions are *components*

- user-level
- minimal unit of spatial isolation

Low-level functions are components

- scheduling
- memory mapping
- I/O processing

Threads orthogonal to components

- thread migration
- concurrent/parallel components

Components interact via *invocation* of exported function

- contractually specified interfaces
- function call semantics

System = Components + Composition

Composition

- complex behavior from simple(ish) pieces
- gluing components together $\rightarrow$ raise level of abstraction

Complex functionality from simple pieces...sound familiar?

Hint: Thompson & Ritchie

System = Components + Composition

Composition

- complex behavior from simple(ish) pieces
- gluing components together $\rightarrow$ raise level of abstraction

Complex functionality from simple pieces...sound familiar?

Hint: Thompson & Ritchie

```
wget -O - www.ecrts.org | grep ‘‘ospert’’ | wc -l
```

System = Components + Composition

Composition

- complex behavior from simple(ish) pieces
- gluing components together $\rightarrow$ raise level of abstraction

Complex functionality from simple pieces...sound familiar?

Hint: Thompson & Ritchie

```
wget -O - www.ecrts.org | grep ‘‘ospert’’ | wc -l
```

```
wget = c "bin/wget" "-O - www.ecrts.org"
grep = c "bin/grep" "ospert"
wc   = c "bin/wc" "-l"
sys  = deps [ (cat, [grep, POSIX]),
              (grep, [wc, POSIX]) ]
```

System = Components + Composition

Composition

- comple
- gluing

bstraction

Complex fu

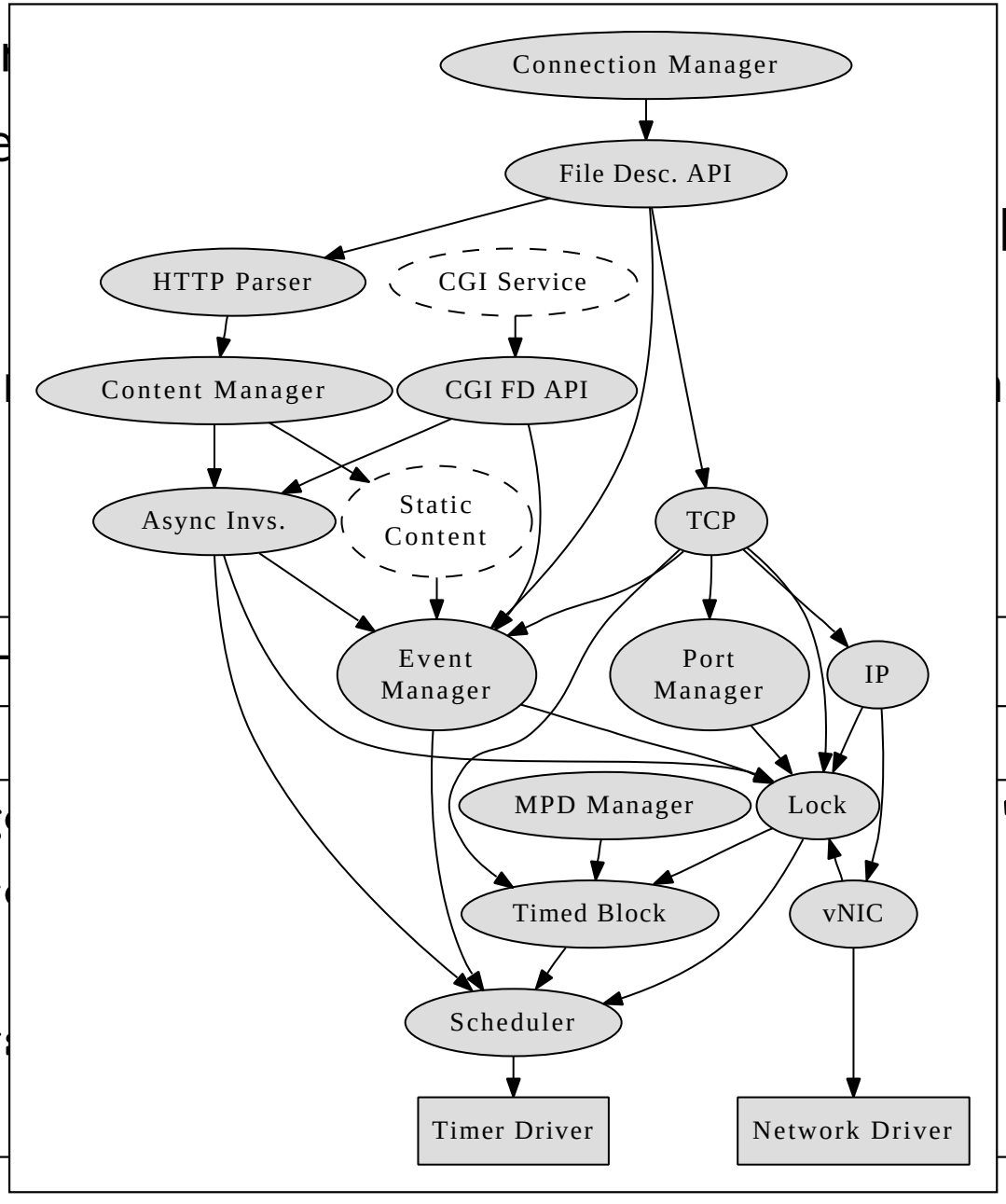
miliar?

```
wget -O -
```

```
' | wc -l
```

```
wg
gr
wc
sy
```

"



System = Components + Composition

Composition

- comple
- gluing

bstraction

Complex fu

miliar?

Challenges:

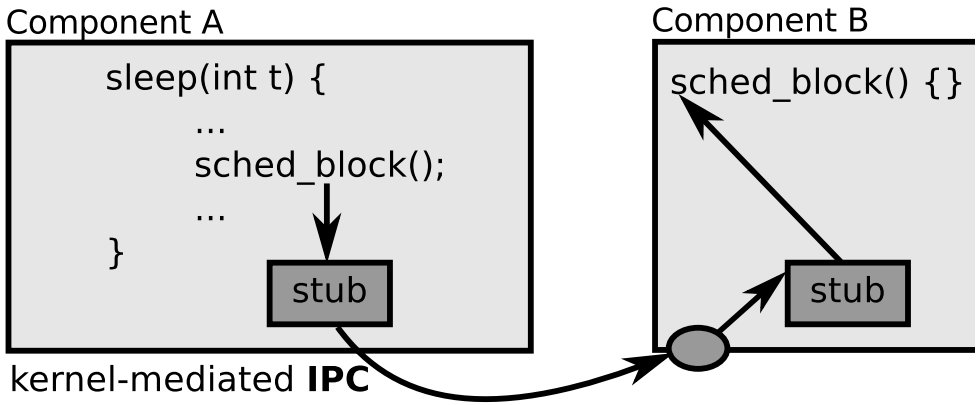
- end-to-end predictability
- dependent-task structure to mirror components?
- trade between component concurrency, and memory

```
wget -O - www.ecrts.org | grep ‘‘ospert’’ | wc -l
```

```
wget = c "bin/wget" "-O - www.ecrts.org"
grep = c "bin/grep" "ospert"
wc   = c "bin/wc" "-l"
sys  = deps [ (cat, [grep, POSIX]),
              (grep, [wc, POSIX]) ]
```

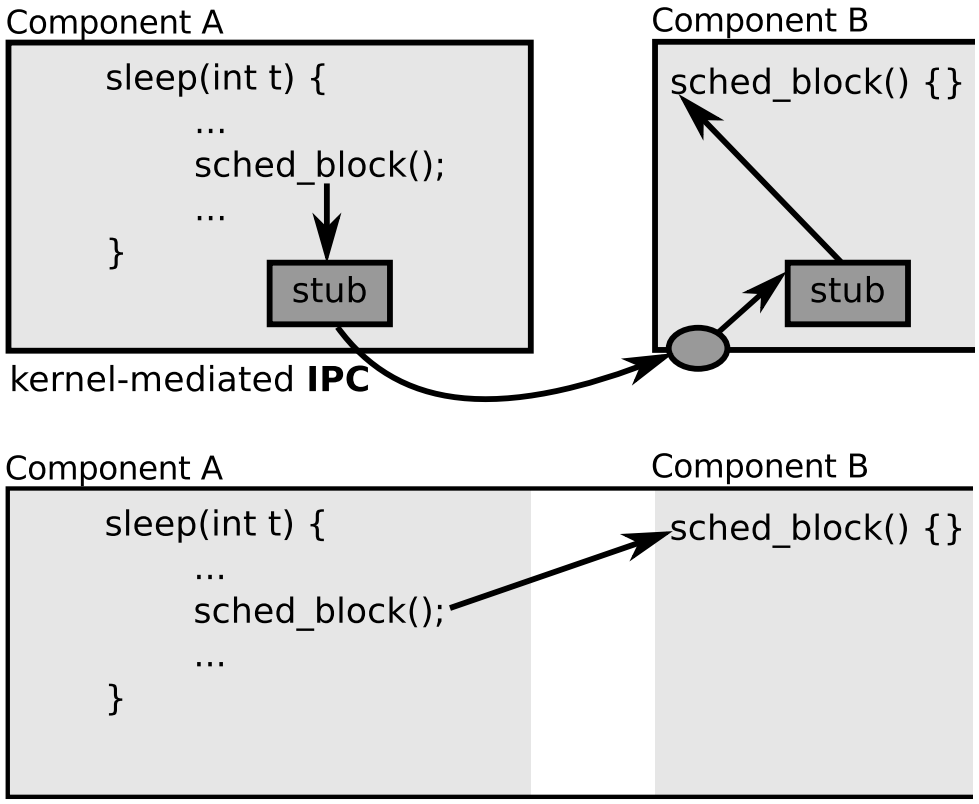
But people understand components...what else?

All problems can be solved by another level of indirection. – Dijkstra



But people understand components...what else?

All problems can be solved by another level of indirection. – Dijkstra



Mutable Protection Domains

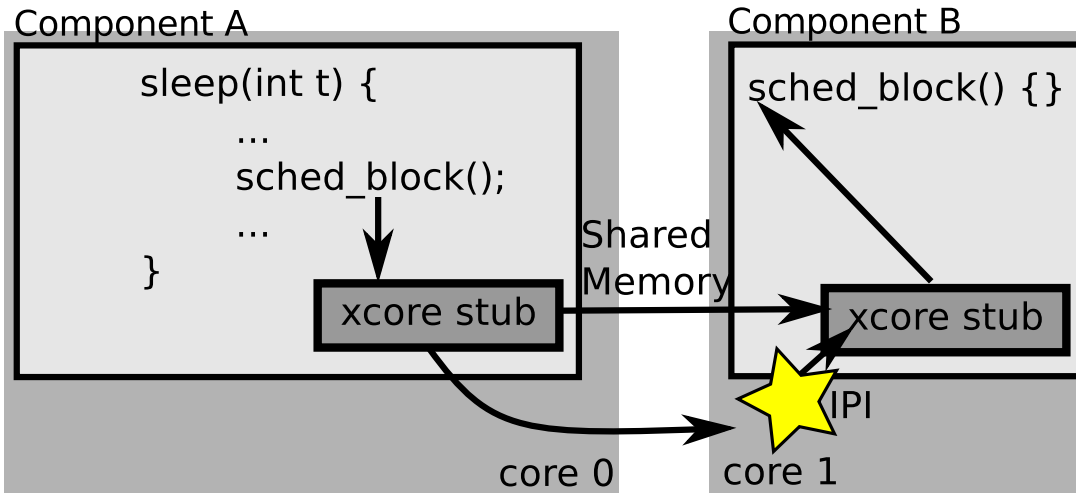
- generalizes other system structures (μkern, exokern, ..)

Predictable Parallel Computation

Parallel systems are here, what do we do with them?

- Inter-task parallelism: simple until
  - shared resources
  - schedulability: partitioned + bin-packing
- Intra-task parallelism:
  - fork/join (OpenMP) schedulability
  - general abstractions + mechanisms for parallelism
  - harness hidden parallelism in concurrent systems
    think: `wget www.ecrts.org& wget www.rtss.org&`

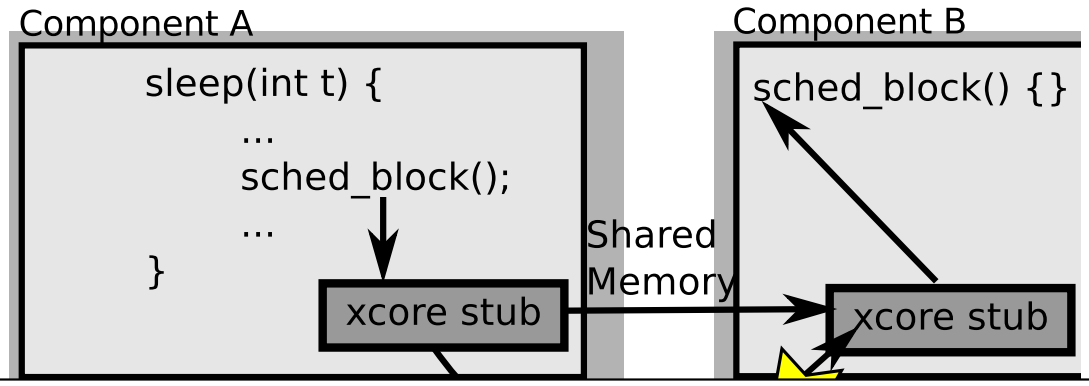
Many-core Composite: MC^2



Inter-component parallelism:

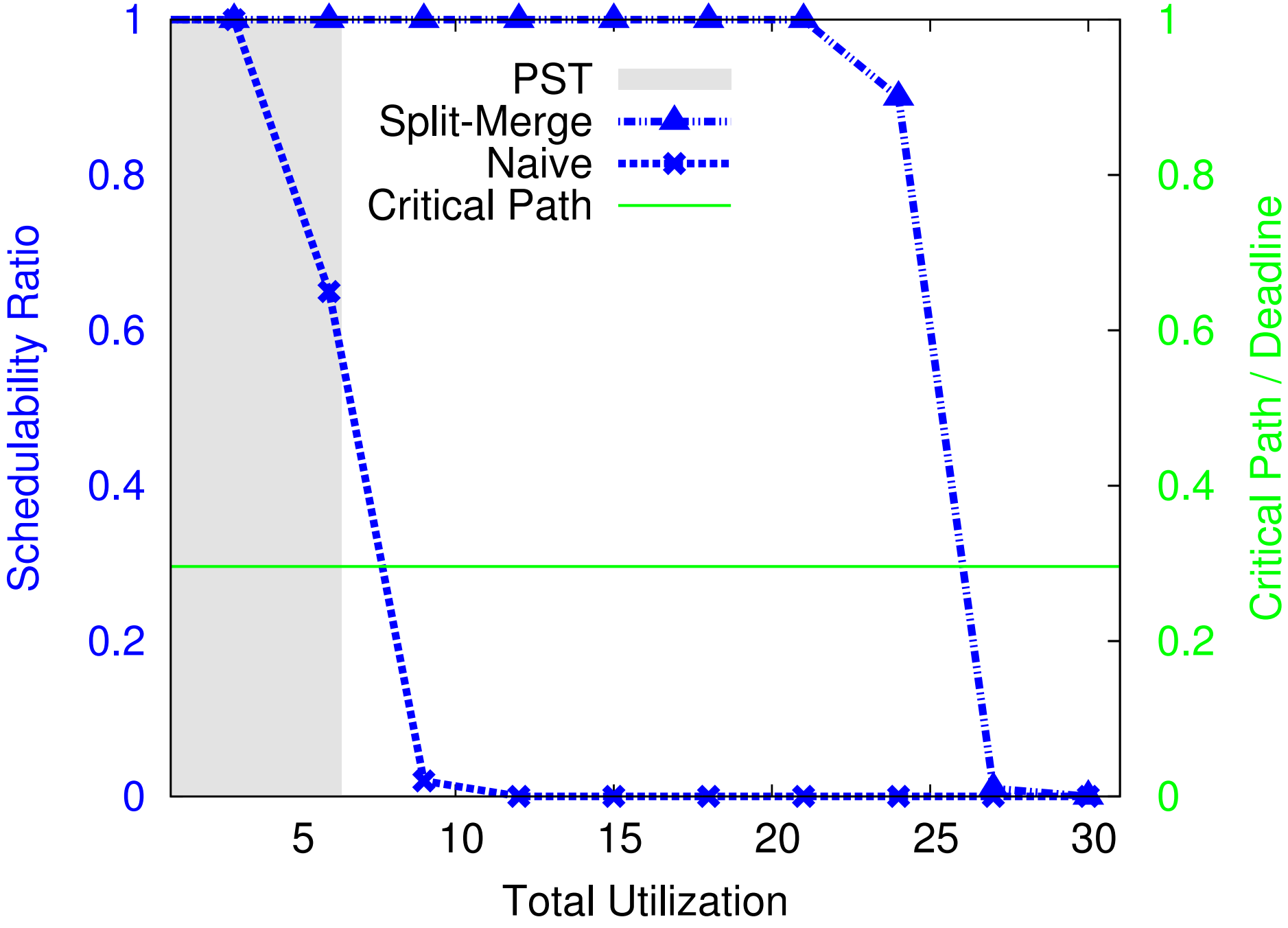
- bin-packing overheads for partitioned systems
- *cut* a task across cores
  - synchronous communication across cores
- specialized mechanisms for cross-core thread activation
  - intra-component: 4x faster than Linux (WC)
  - inter-component: harness non-blocking, async APIs

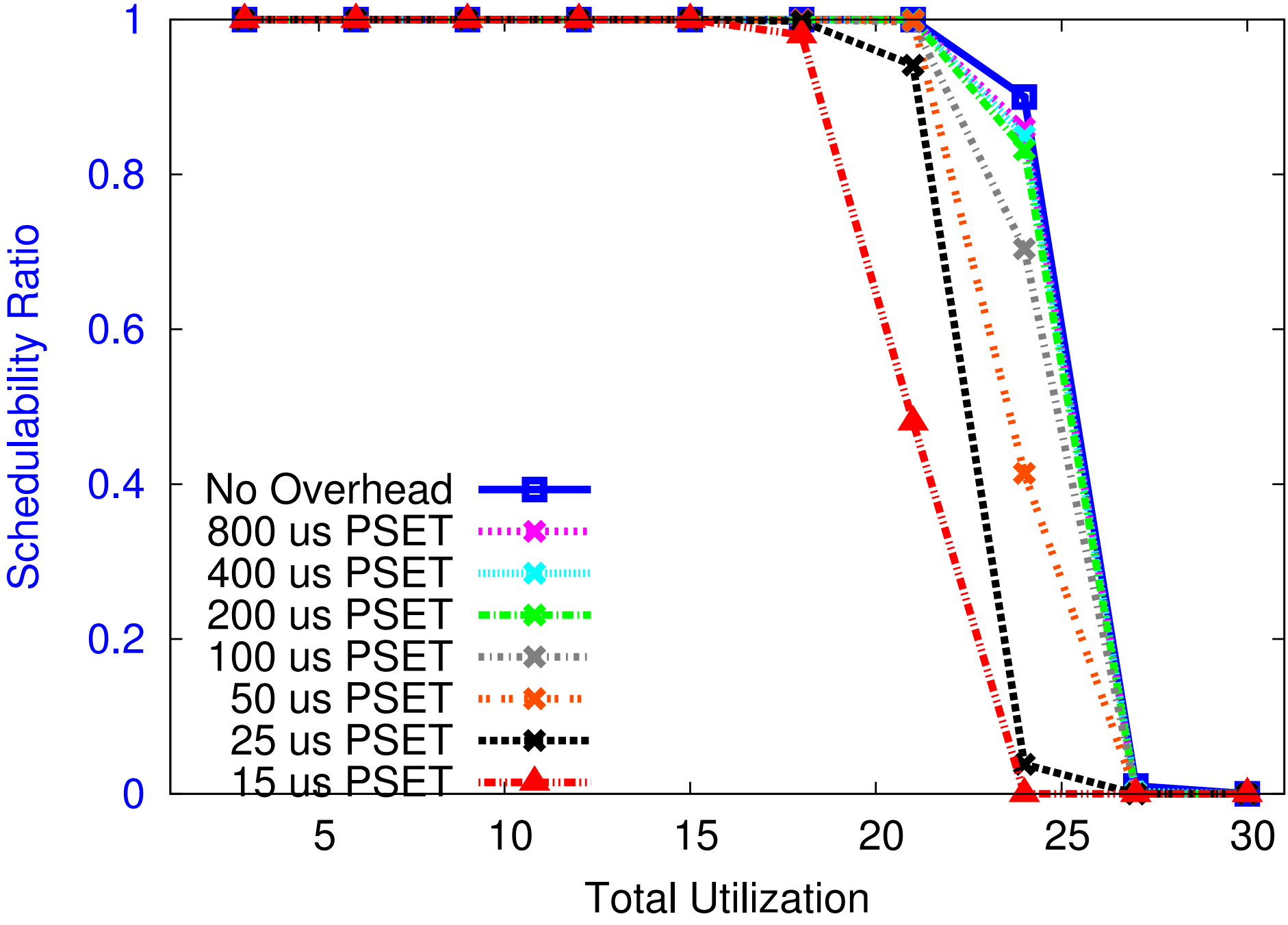
Many-core Composite: MC^2



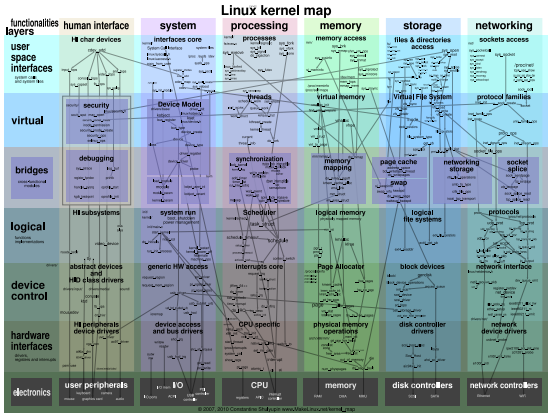
Pair this with:
– a smart assignment algorithm, and
– optimized holistic analysis to analyze schedulability.

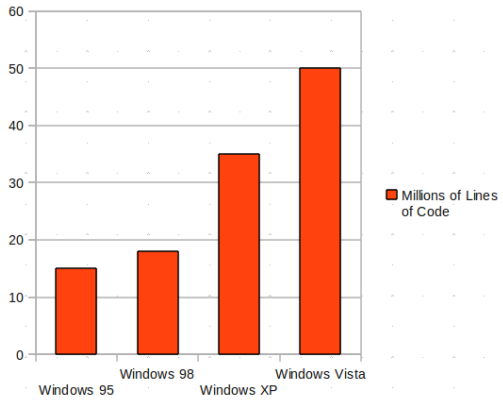
- bin-packing overheads for partitioned systems
- *cut* a task across cores
  - synchronous communication across cores
- specialized mechanisms for cross-core thread activation
  - intra-component: 4x faster than Linux (WC)
  - inter-component: harness non-blocking, async APIs


PST
Split-Merge
Naive
Critical Path
Schedulability Ratio
Critical Path / Deadline
Total Utilization
0
0.2
0.4
0.6
0.8
1
5
10
15
20
25
30


Schedulability Ratio
1
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
Total Utilization
No Overhead
800 us PSET
400 us PSET
200 us PSET
100 us PSET
50 us PSET
25 us PSET
15 us PSET

Transparent, System-Provided, Fault Tolerance

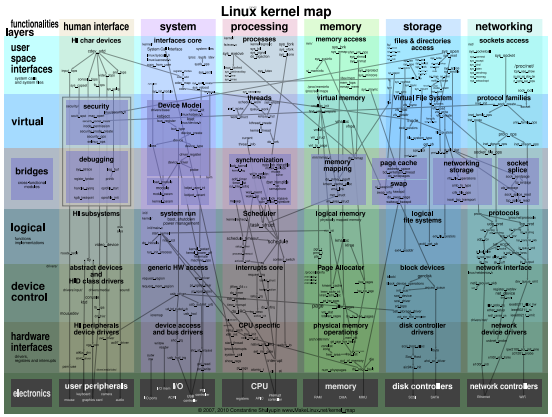


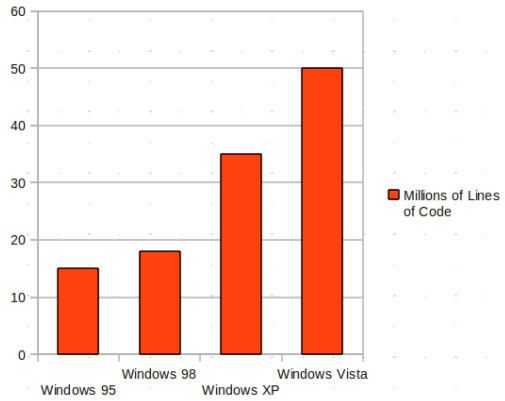


Decreasing process sizes

- \+ faster
- \+ less power
- \+ smaller
- – increased vulnerability to HW transient faults
- – 65% of HW faults corrupt OS state

Transparent, System-Provided, Fault Tolerance





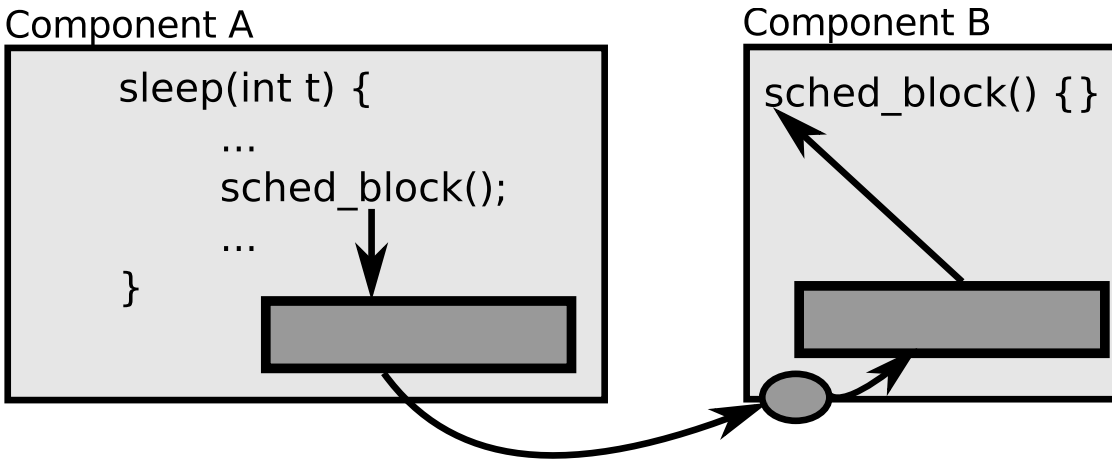
Decreasing process sizes

- \+ faster
- \+ less power
- \+ smaller
- – increased vulnerability to HW transient faults
- – 65% of HW faults corrupt OS state

Can we provide fault tolerance

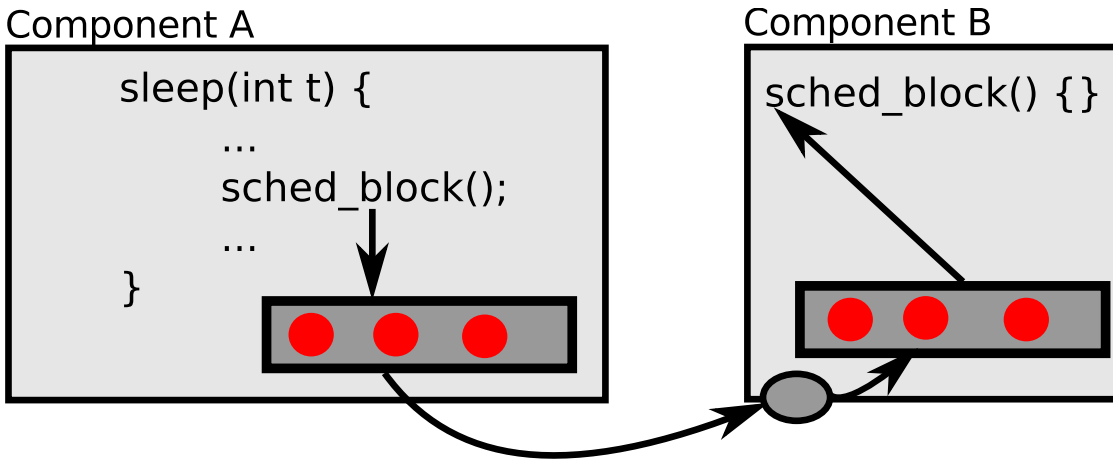
- even for the lowest-level components?
- predictably and efficiently?

Computational Crash Cart: C^3



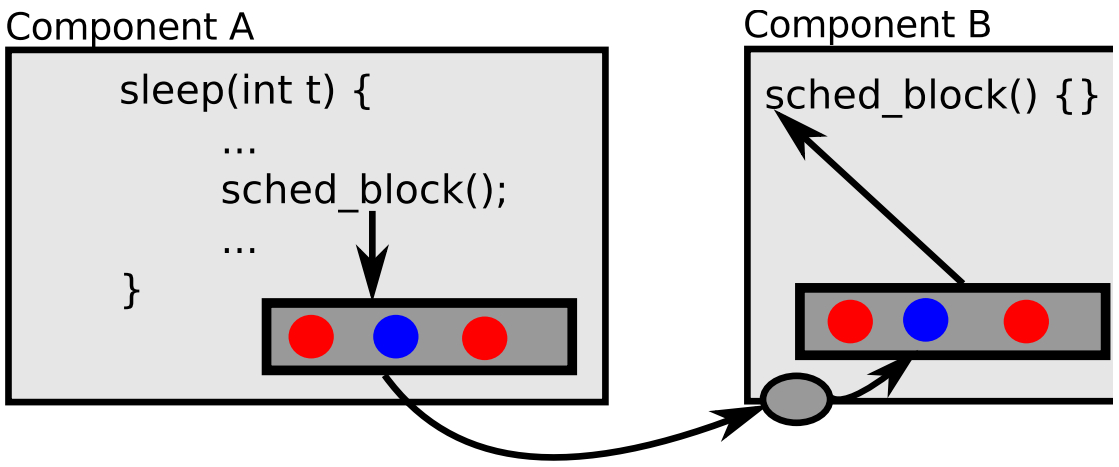
1. interpose on communication between components
2. track state of each "shared" object
   - file, thread, lock, ...
3. fault in server!
4. μ-reboot component
5. rebuild state via functions in interface

Computational Crash Cart: C^3



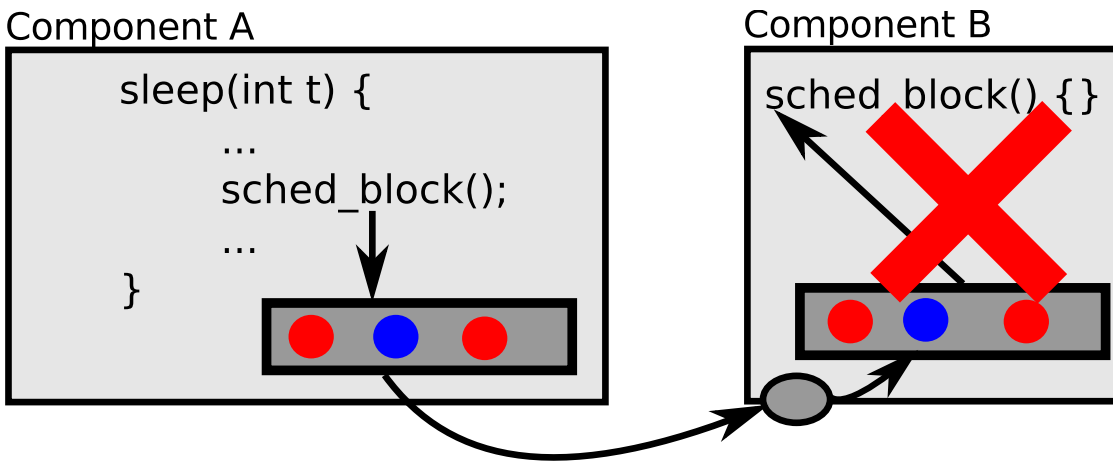
1. interpose on communication between components
2. track state of each "shared" object
   - file, thread, lock, ...
3. fault in server!
4. μ-reboot component
5. rebuild state via functions in interface

Computational Crash Cart: C^3



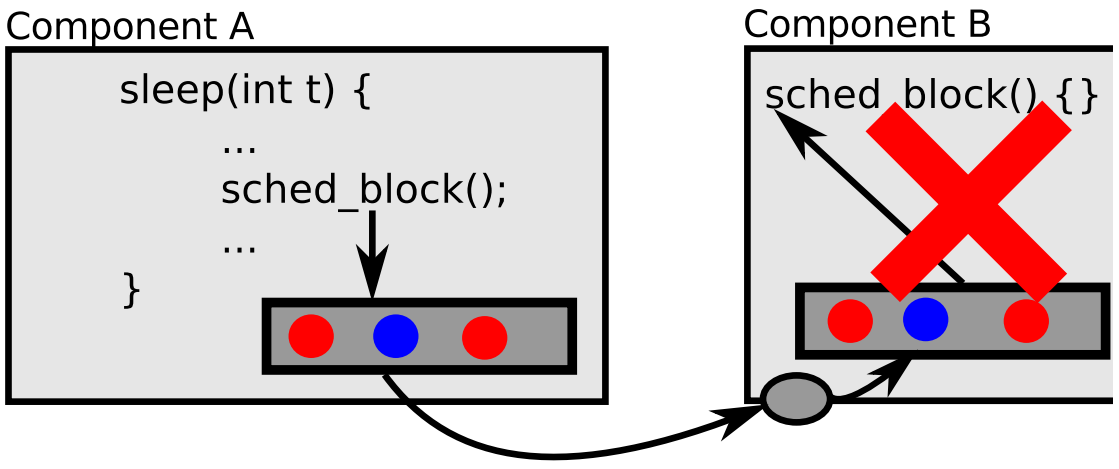
1. interpose on communication between components
2. track state of each "shared" object
   - file, thread, lock, ...
3. fault in server!
4. μ-reboot component
5. rebuild state via functions in interface

Computational Crash Cart: C^3



1. interpose on communication between components
2. track state of each "shared" object
   - file, thread, lock, ...
3. fault in server!
4. μ-reboot component
5. rebuild state via functions in interface

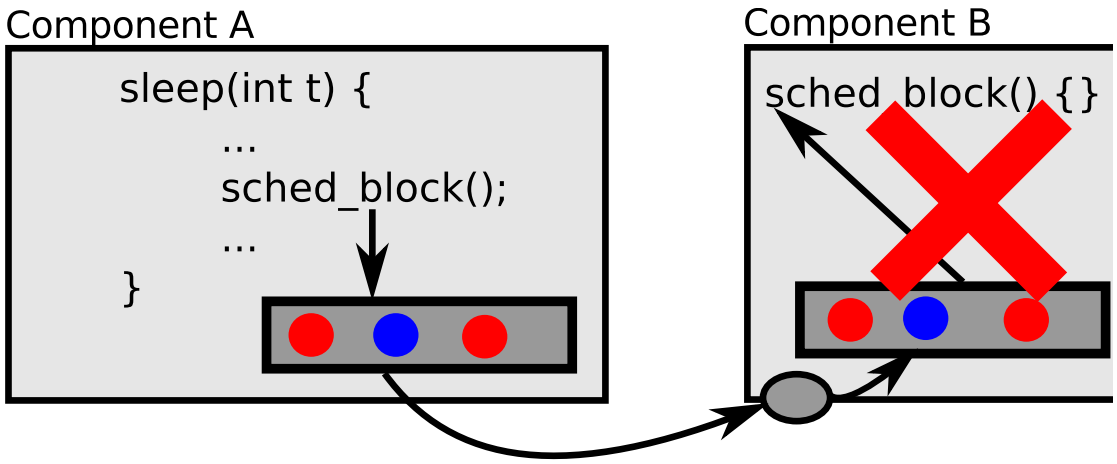
Computational Crash Cart: C^3



Recovery affects timing of multiple threads

- performed on-demand by thread using object
- rebuild objects at proper priority
- avoid *recovery inversion*

Computational Crash Cart: C^3

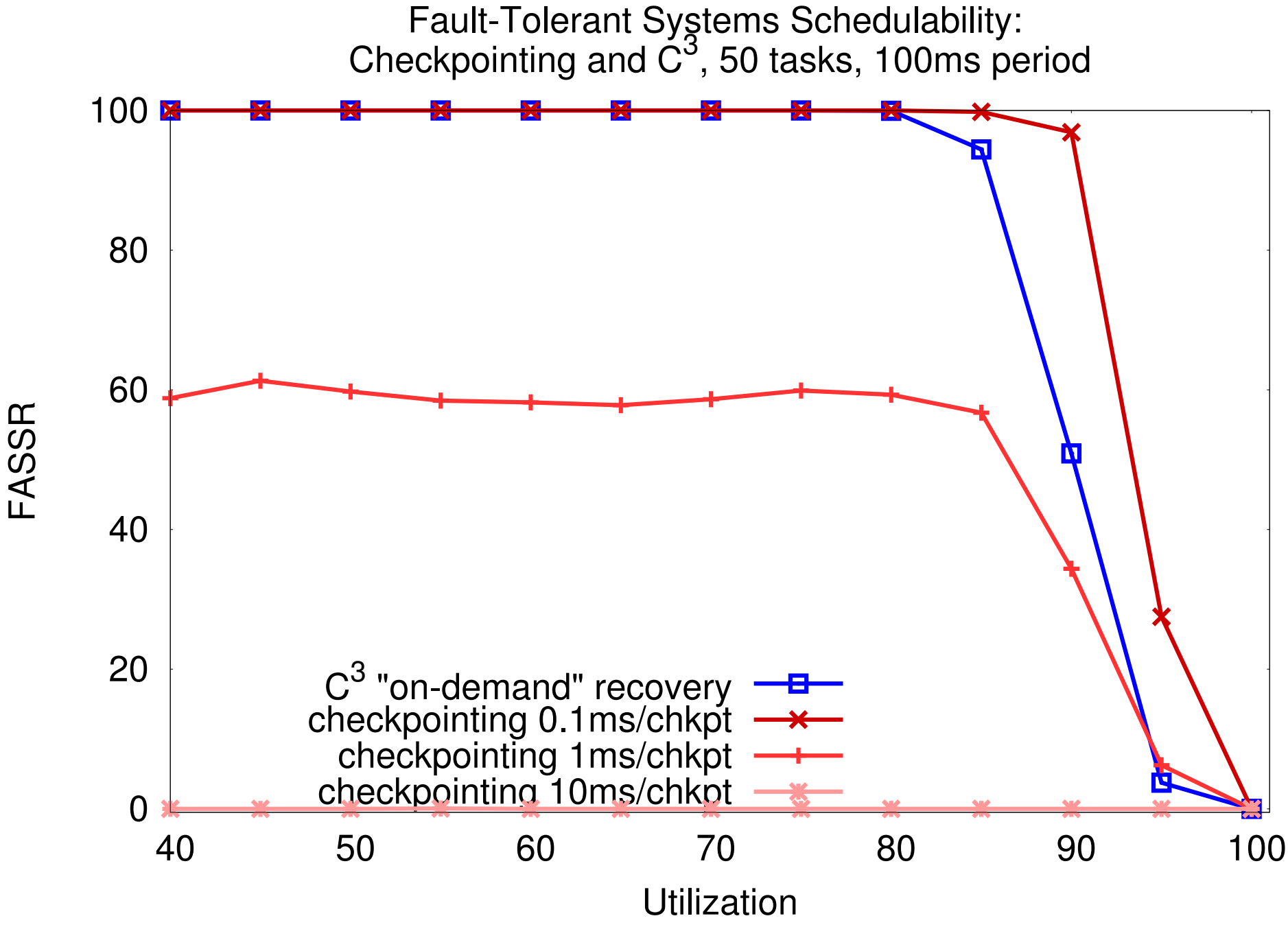


C^3: Efficient, system-wide fault tolerance

- recovers 100% injected faults (scheduler, memmgr, fs)
- μ-reboot in $< 20\mu$-sec
- rebuild object: $< 5\mu$-sec

Versus checkpointing

- CRIU: 10ms, Xen: 10sec
- C^3 : 0.1ms per MB


Fault-Tolerant Systems Schedulability:
Checkpointing and C3, 50 tasks, 100ms period
FASSR
Utilization
100
80
60
40
20
0
40
50
60
70
80
90
100
C3 "on-demand" recovery
checkpointing 0.1ms/chkpt
checkpointing 1ms/chkpt
checkpointing 10ms/chkpt

The State of Composite is...

...in progress.

- MC^2: Full-system, predictable parallelism
- C^3: Predictable, system-level fault tolerance
- HierOS: hierarchical paravirtualization
  (FreeRTOS done, Linux in-progress)
- IsolOS: separation kernel support
- SecCOS: fine-grained authentication + monitoring
- ...POSIX support (see Rob Pike's polemic)

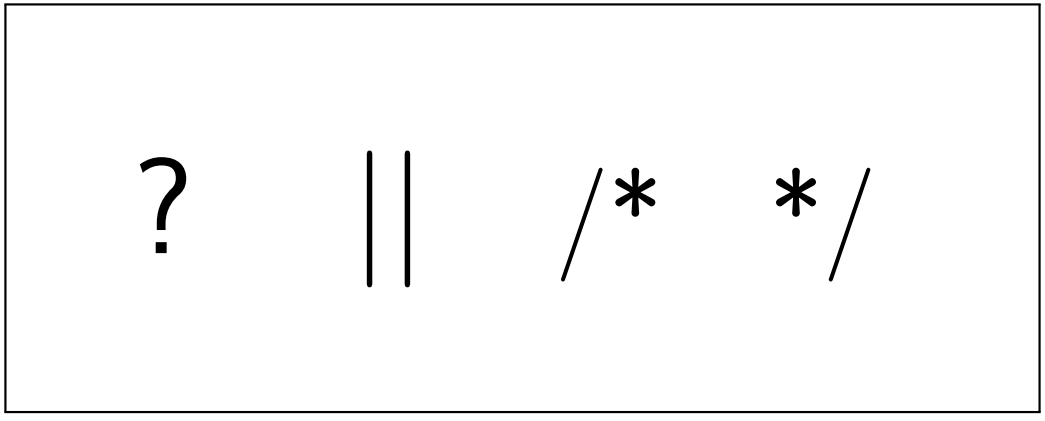
Composite as CBOS:

- configurable to system reqs; as complex as required
- generalizes system structures

Composite as memory isolation + function call indirection

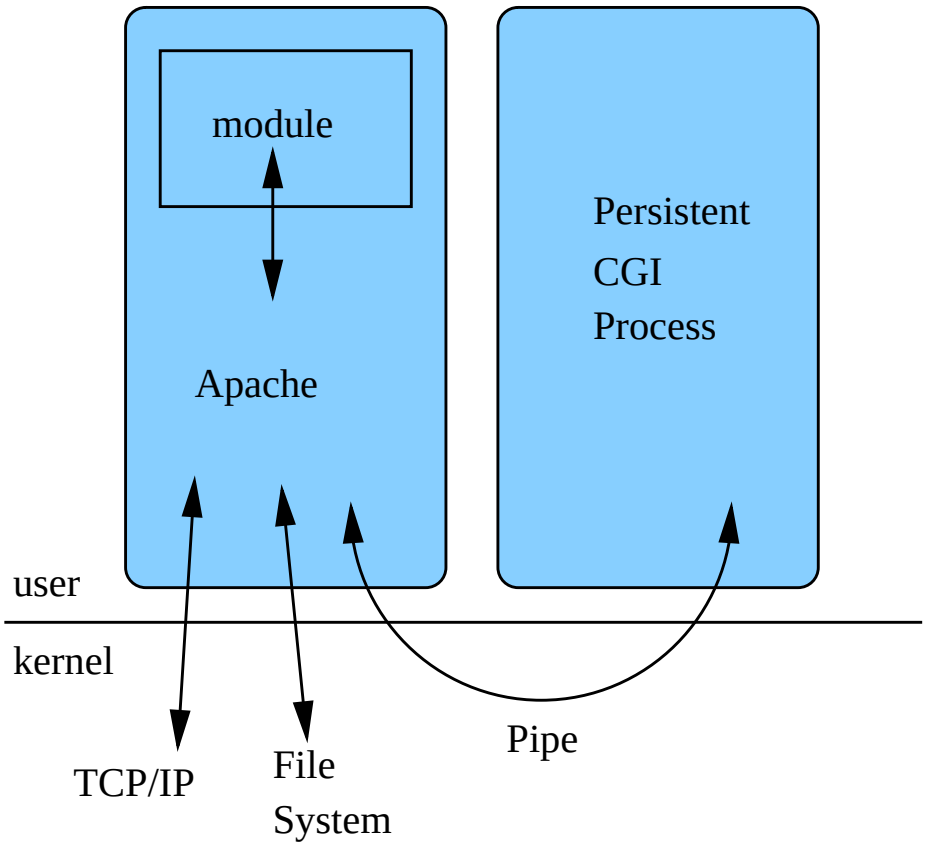
- general, transparent parallelism
- system-level fault tolerance

Thank You!



composite.seas.gwu.edu

Comparison Case: Apache Web-Server, Linux

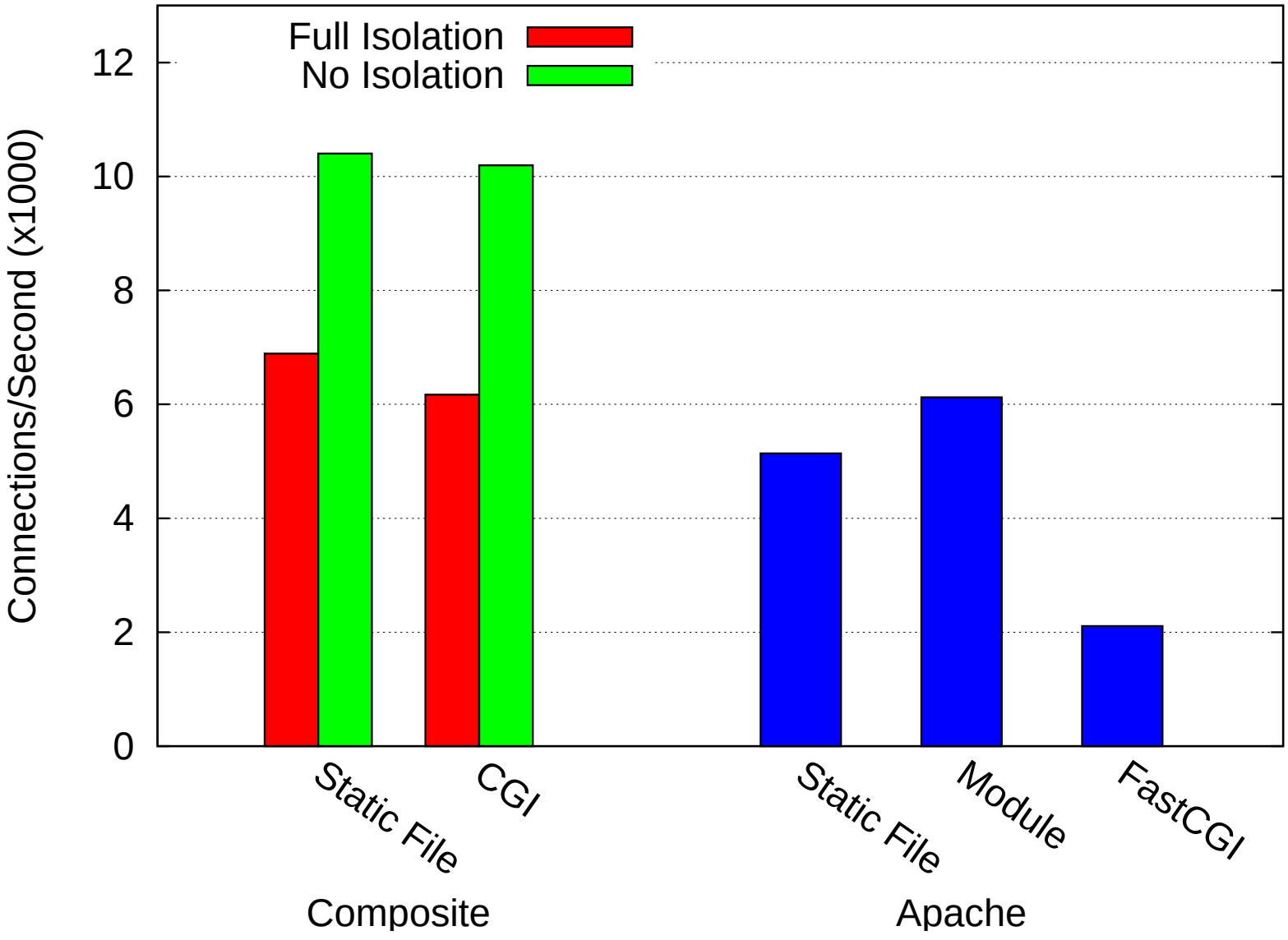


Apache provides multiple content sources

Figures to keep in mind:

- Linux CGI communication (pipe RPC): 6.4 μ-sec
- COMPOSITE component communication: 0.67 μ-sec

Apache, Composite Comparison



Resource Management

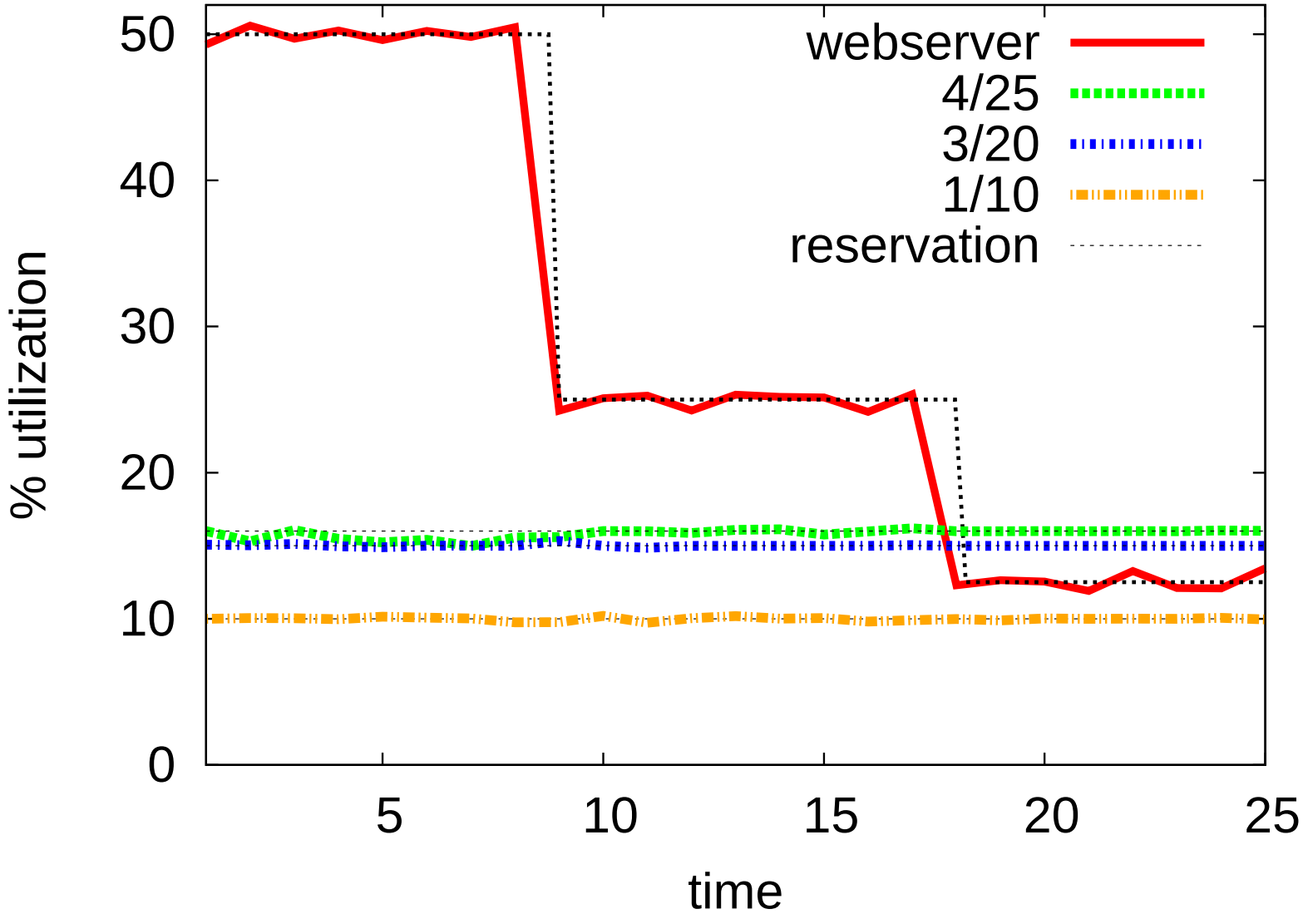
Components configured in the system:

- schedulers
- memory mappers
- I/O managers
- file systems
- networking protocols
- ...

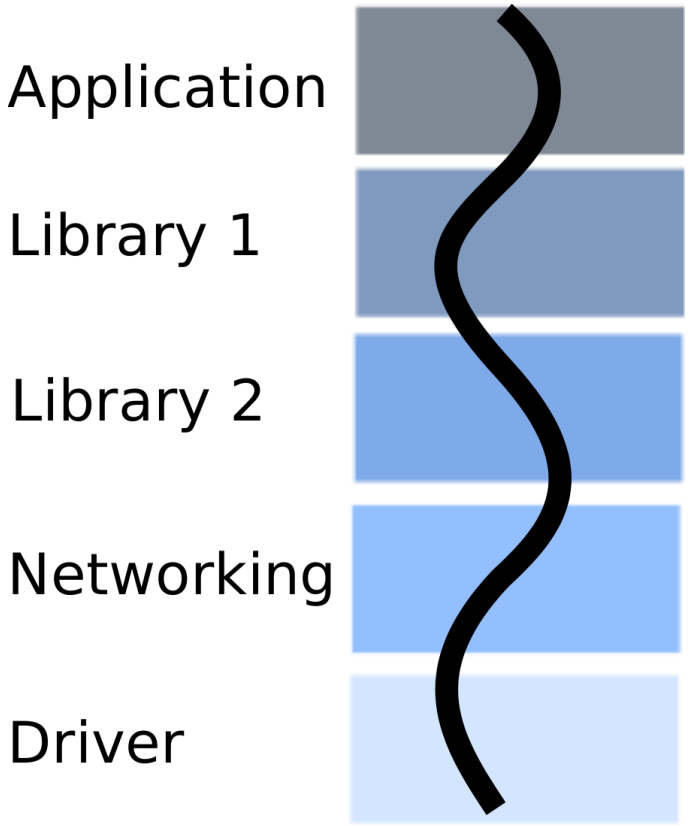
Cost of component resource mgmt? (in μ-seconds)

- Scheduler: thread switch – 0.4 (cos) vs. 0.8 (linux)
- Memory mapping: `mmap` – 2 (cos) vs. 6 (linux)
- I/O: receive packet – 9.69 (cos) vs. 10.3 (linux)

Best Effort Subsystem vs. RT Task Execution



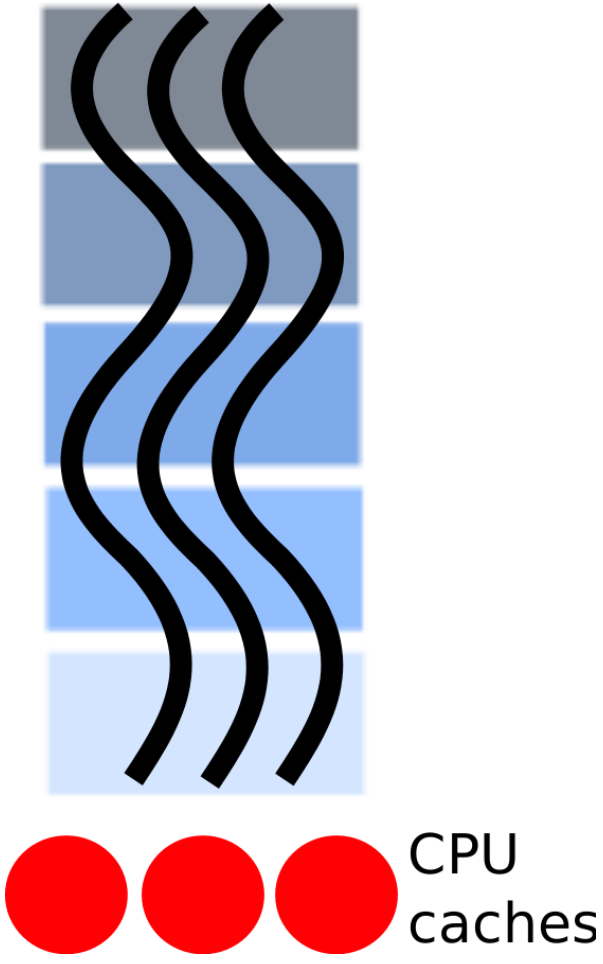
System Management of Parallelism



Traditional model of computation

- thread executes through system layers
- each layer has its own data working set

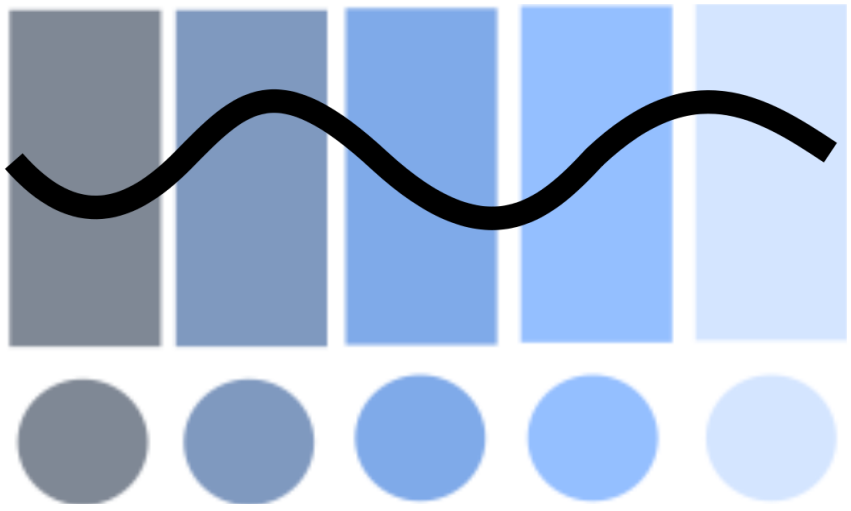
System Management of Parallelism



Traditional model of utilizing parallelism

- thread execute through same layers
- same data working sets in each cache
  $\rightarrow$ inefficient use of caches!

System Management of Parallelism



COMPOSITE w/ invocations spreading computation across cores

- CPU caches specialize around a specific working set
- controlled cache inefficiency
  - factor of 100 performance difference
- control the parallelism of any one component