



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Using Real-Time Linux in Real Life

Klaas van Gend
Senior Solutions & Services Architect
MontaVista Software

montavista™



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Who is Klaas van Gend?

Klaas-the-Geek:

- Started programming age 13
- First encountered Linux 1993
- Software Engineer since 1998
- Lead developer of umtsmon
- Program Committee member for various open source conferences



Klaas-the-Sales-Guy:

- Joined MontaVista as FAE (not sales) 2004
- Was part of European MontaVista Team
- Awarded FAE of the year 2006
- Working in USA until July 1st, 2009



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

History of Linux and Real Time

UNIX = Fair

Preemption in user space

Fixed Overhead / O(1) Scheduler

Robert Love's Kernel Preemption

Ingo Molnar's Voluntary preemption

The RT patch

Paul McKenney's RCU work

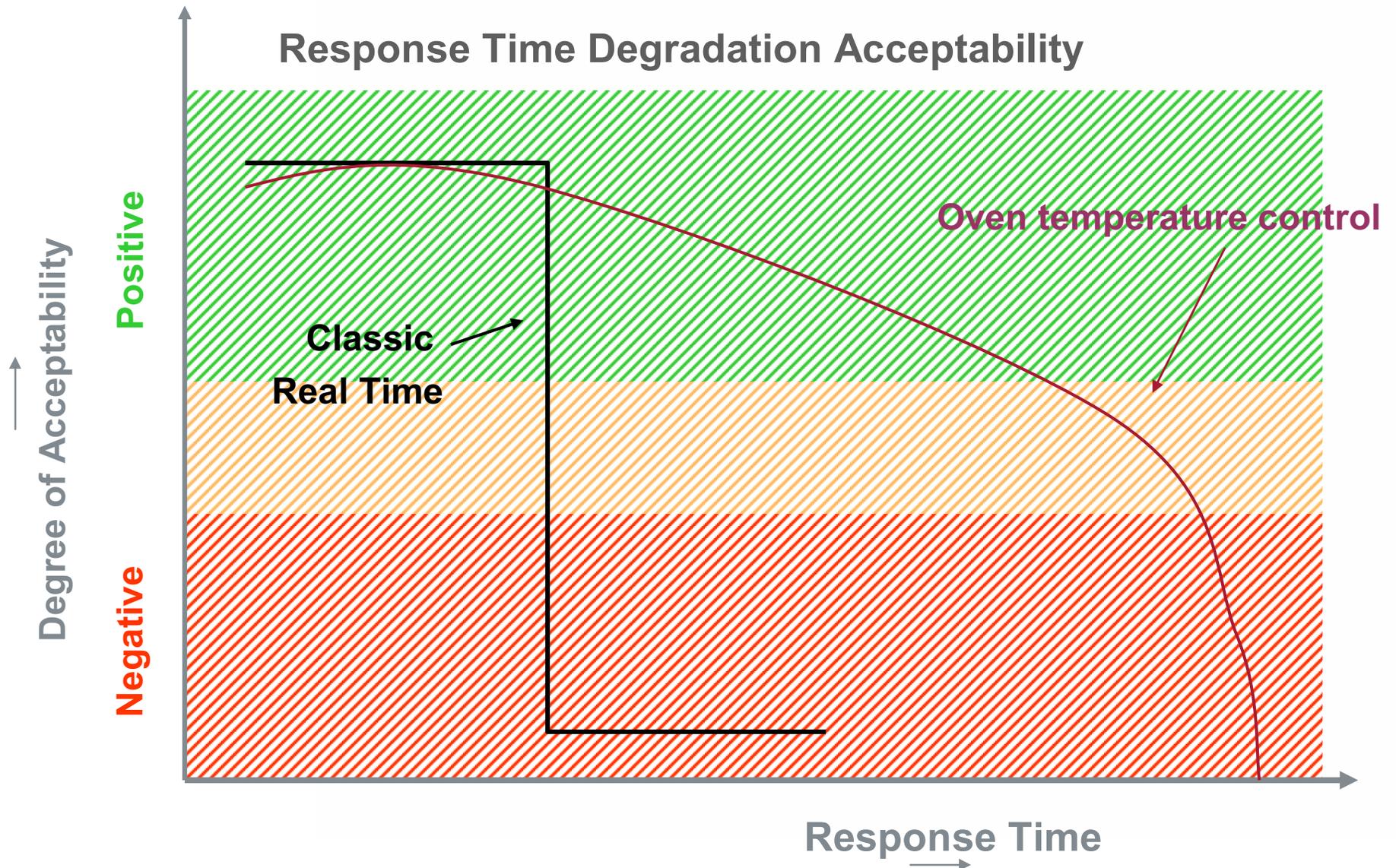
Ingo Molnar's new CFSScheduler

Gregory Haskin's Optimizations/Scheduler work





Two types of Real-Time Expectations





montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Real Time Linux

Main assumption:

The highest priority task goes first

ALWAYS

Thus:

Everything should be pre-emptable

Nothing should keep higher priority things from executing





montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Key Elements of Real-Time Linux

Making Linux Real-time required addressing:

- Minimized interrupt disable times

- Interrupt handling via schedulable threads

- Fully pre-emptable kernel

 - Short critical sections

- Perform synchronization via mutexes (not spin locks)

 - Allows involuntary pre-emption

- Mutex support for priority inheritance

- High Resolution timers

- Minimize number of mutexes / critical sections

- Optimizing scheduler decisions



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Sleeping Spinlocks

Original Linux **UP** Spinlock:

IRQ disable on lock – nothing else can interrupt

Not RT friendly

Original Linux **SMP** Spinlock:

Spinning (busy wait)

Not performance friendly



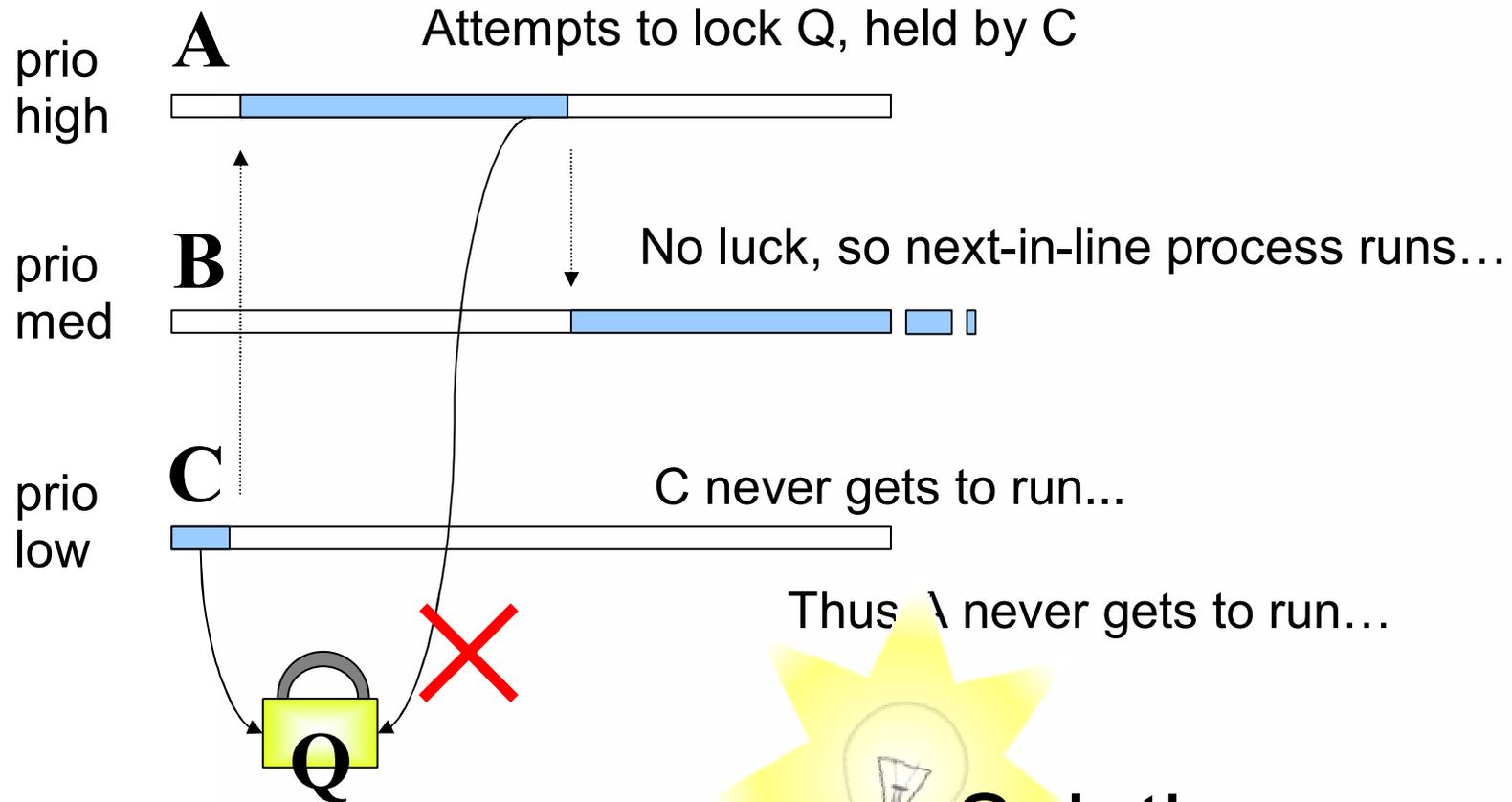
Solution:

“Sleeping Spinlock”





Problem: Priority Inversion



 Solution:
“Priority Inheritance”



Robust Mutexes

Problem:

Inter *process* semaphores (“named ~”)

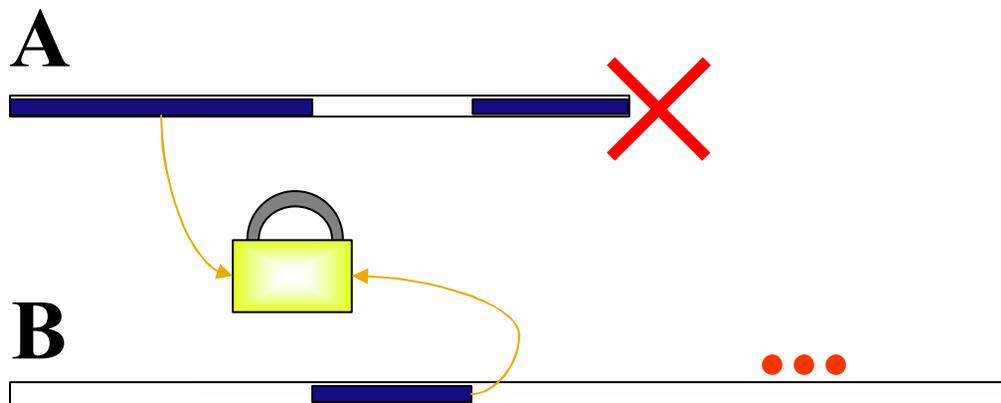
Process **A** holds semaphore and dies

Process **B** blocks on the same semaphore

On regular Linux: ***mutex locked forever***

Thus waiting process **B** held forever

...until reboot



Solution:
“Robust
Mutex”





montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Priority Queues

Problem:

1000 processes waiting for a locked mutex

Mutex gets unlocked – ***who will go first?***

On regular Linux, the first waiting process ‘gets’ the mutex

On RT Linux, the *highest priority* process should wake up and get the lock



Solution:

“Priority Queues”

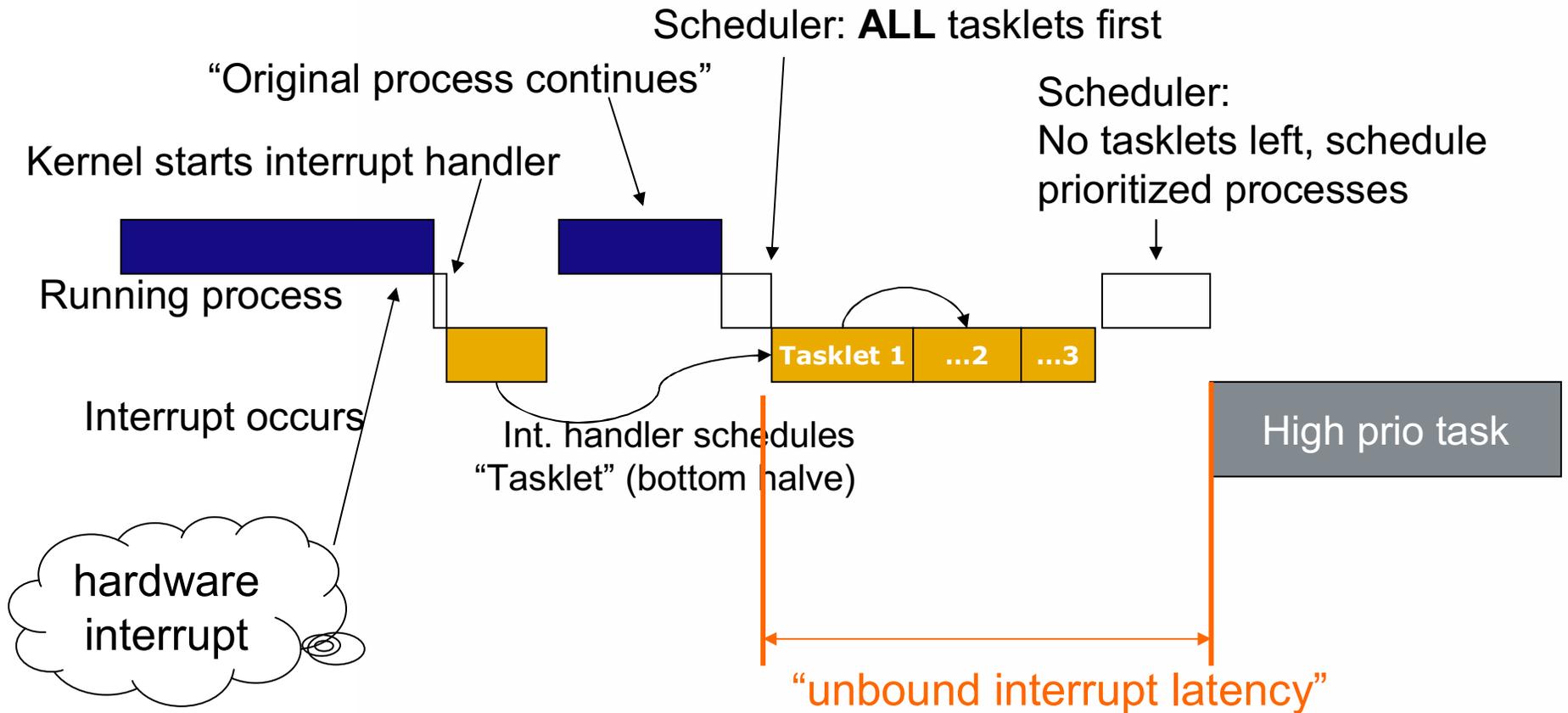


**Real Time
is NOT fair,
remember?**





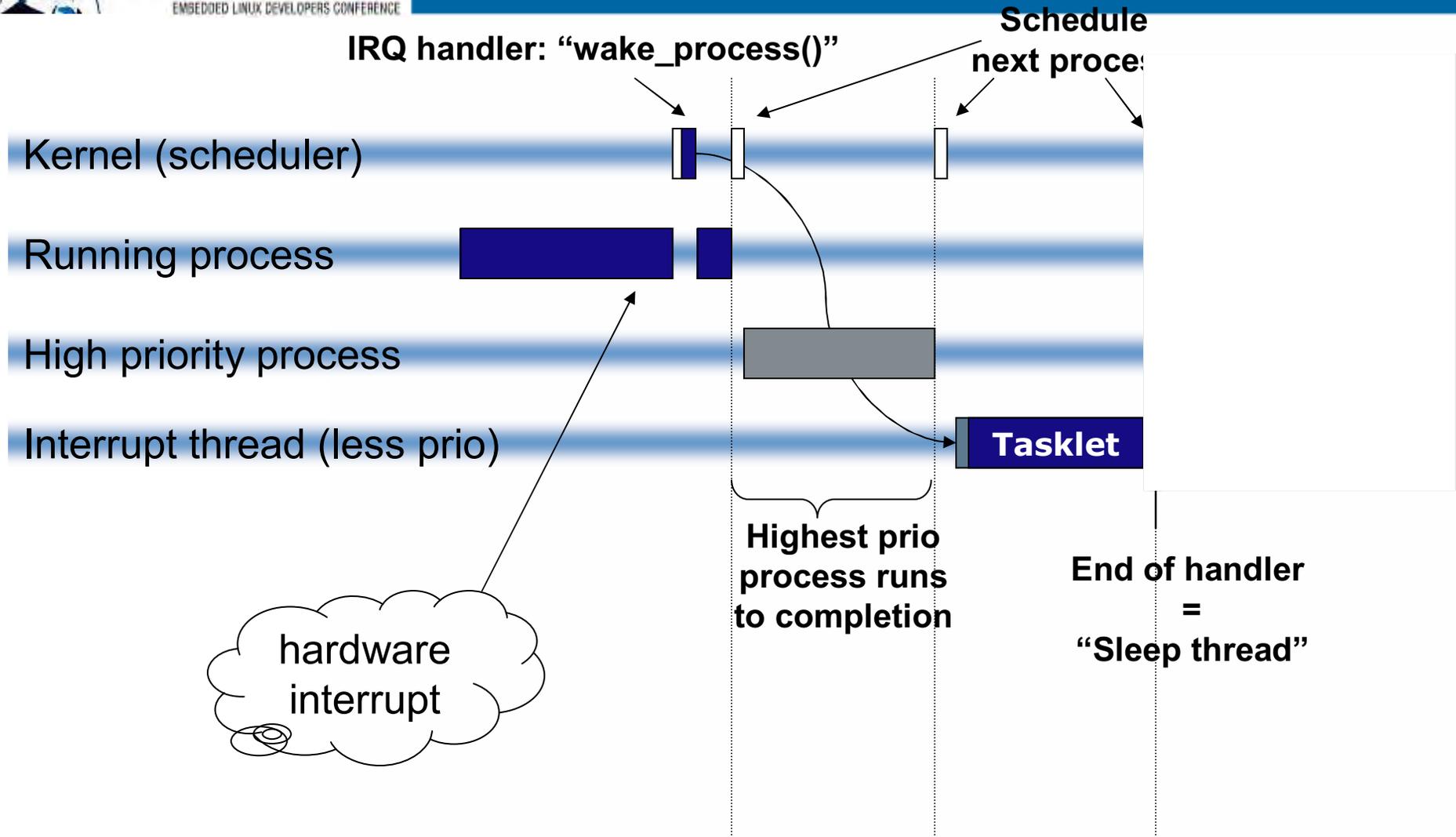
What's wrong with the standard IRQ mechanism?



Solution:
"Threaded IRQs"

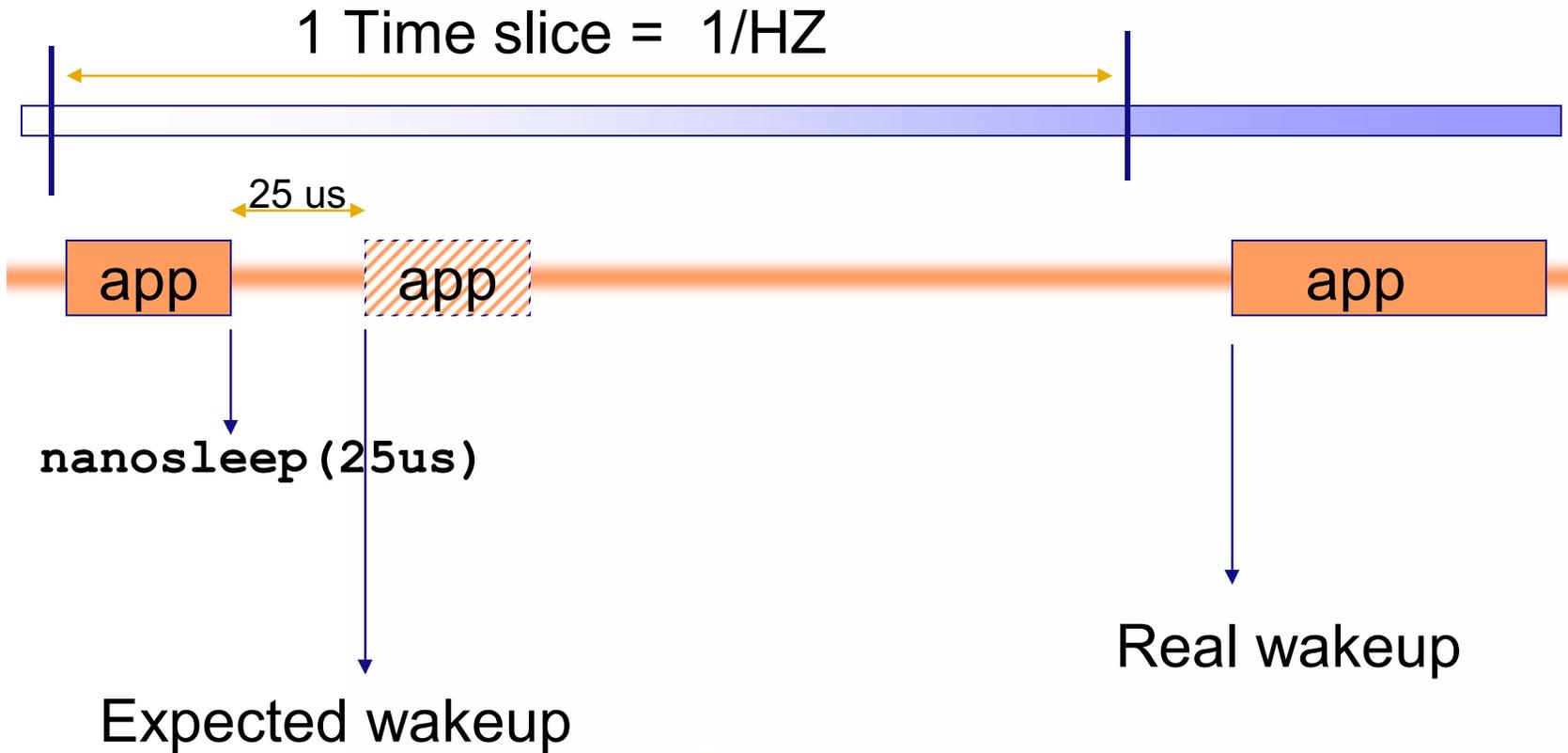


RT-patch Thread Context Interrupt Handlers





Time sliced based timing



Solution:
“Hi-res Timers”



Key Elements of Real-Time Linux (revisited)

Making Linux Real-time required addressing:	Finished?	Mainline?
Minimized interrupt disable times BKL still present !!!	Ongoing	Some
Interrupt handling via schedulable threads Not acceptable to all drivers	Done	NO
Fully pre-emptable kernel Short critical sections	Done	NO
Perform synchronization via mutexes (not spin locks) Allows involuntary pre-emption	Done	Partly
Mutex support for priority inheritance / queueing		
High Resolution timers	Done	Yes
Minimize number of mutexes / critical sections	Ongoing	Some
Optimizing scheduler decisions New Scheduler (CFS) in mainline	Ongoing	Some



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Some Results

i.e. Benchmarks

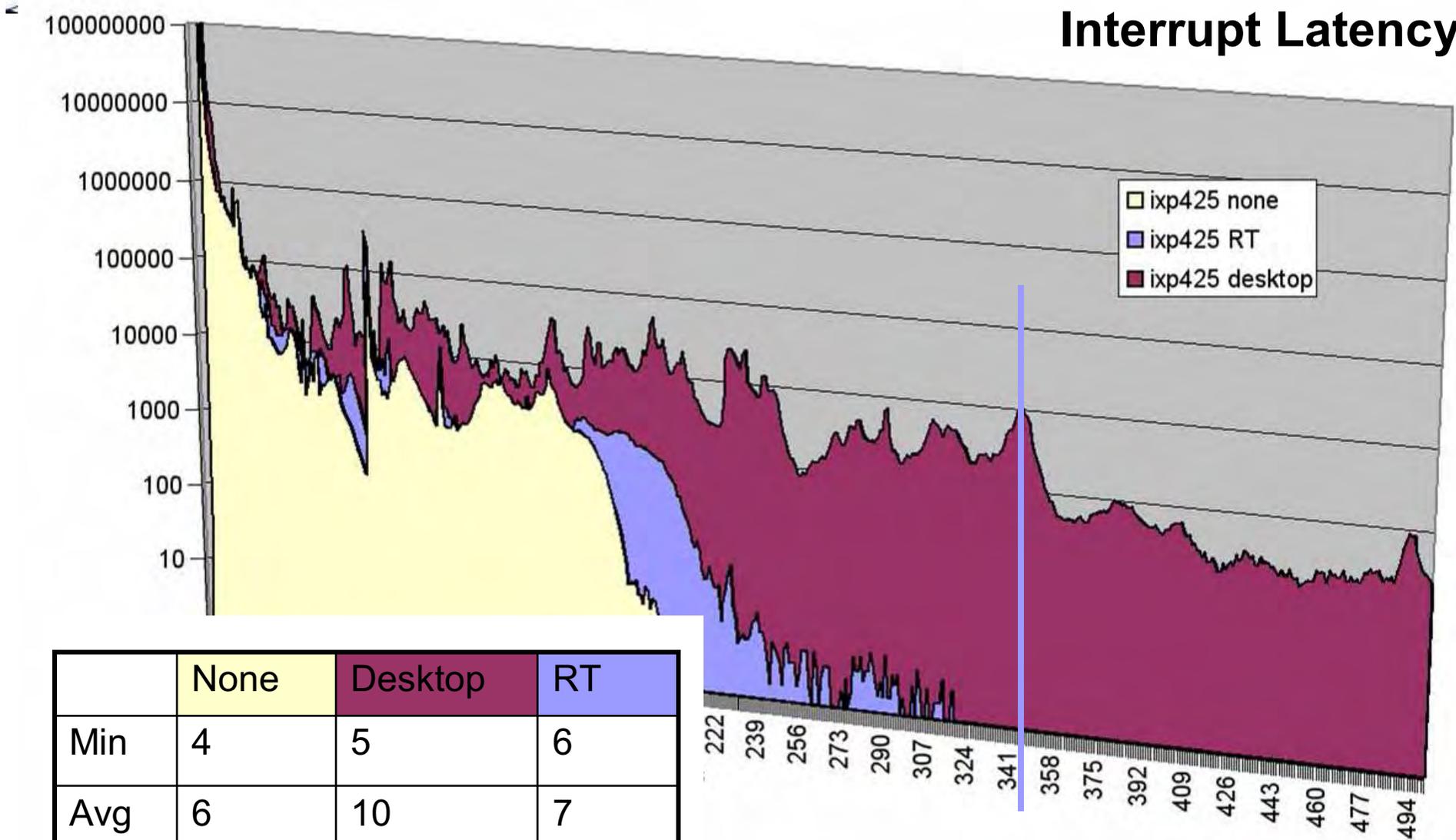
i.e. Synthetic



montavista
VISION

Intel IXP425 @ xxx Mhz, 2.6.18+

Interrupt Latency



	None	Desktop	RT
Min	4	5	6
Avg	6	10	7
Max	9797	2679	349

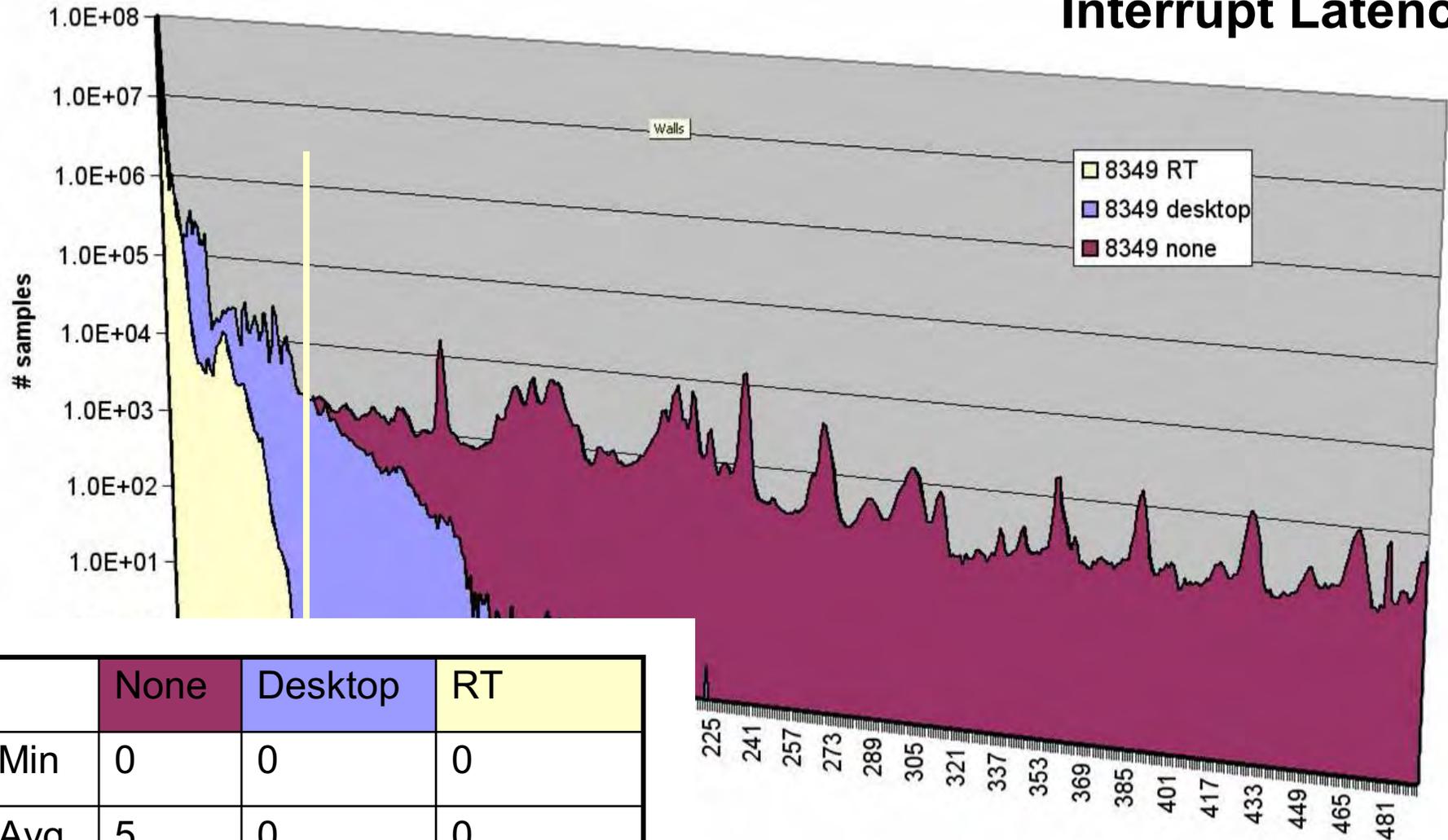
RT Limit



montavista
VISION
2008

FreeScale 8349 mITX @xxxMHz, 2.6.18+

Interrupt Latency



	None	Desktop	RT
Min	0	0	0
Avg	5	0	0
Max	3968	1604	53



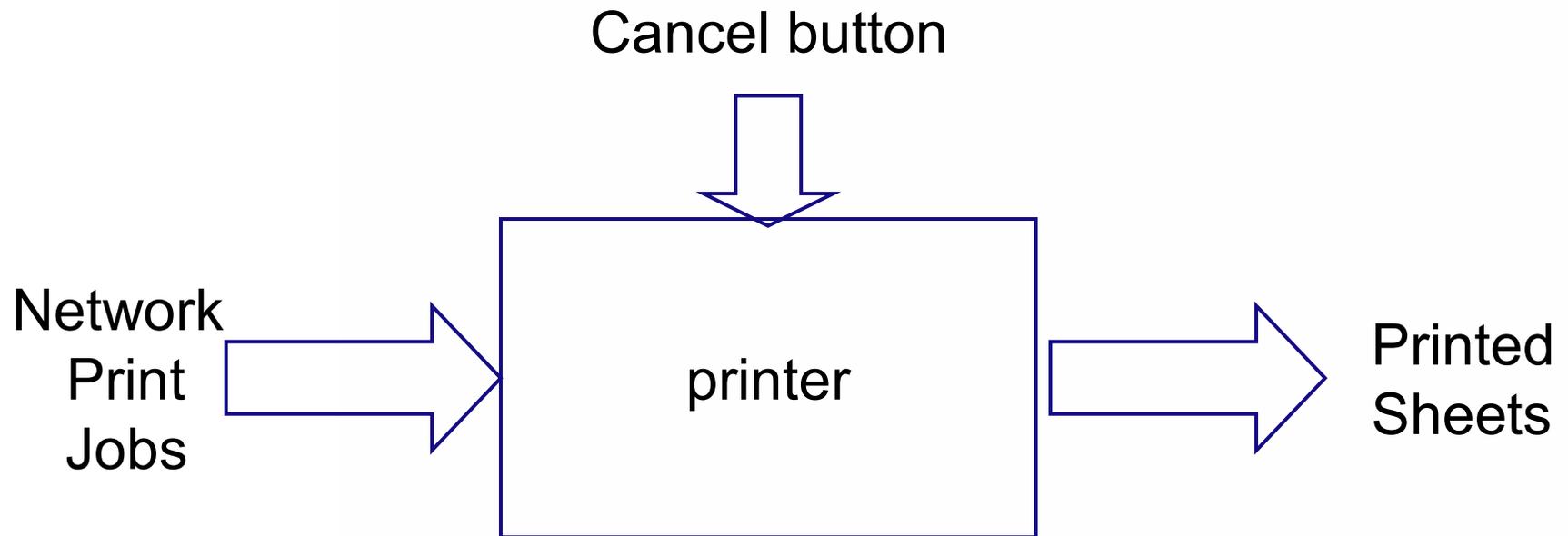
montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

What about “In Real Life”?

Or: Thanks for all the theory,
but where’s the applicability?

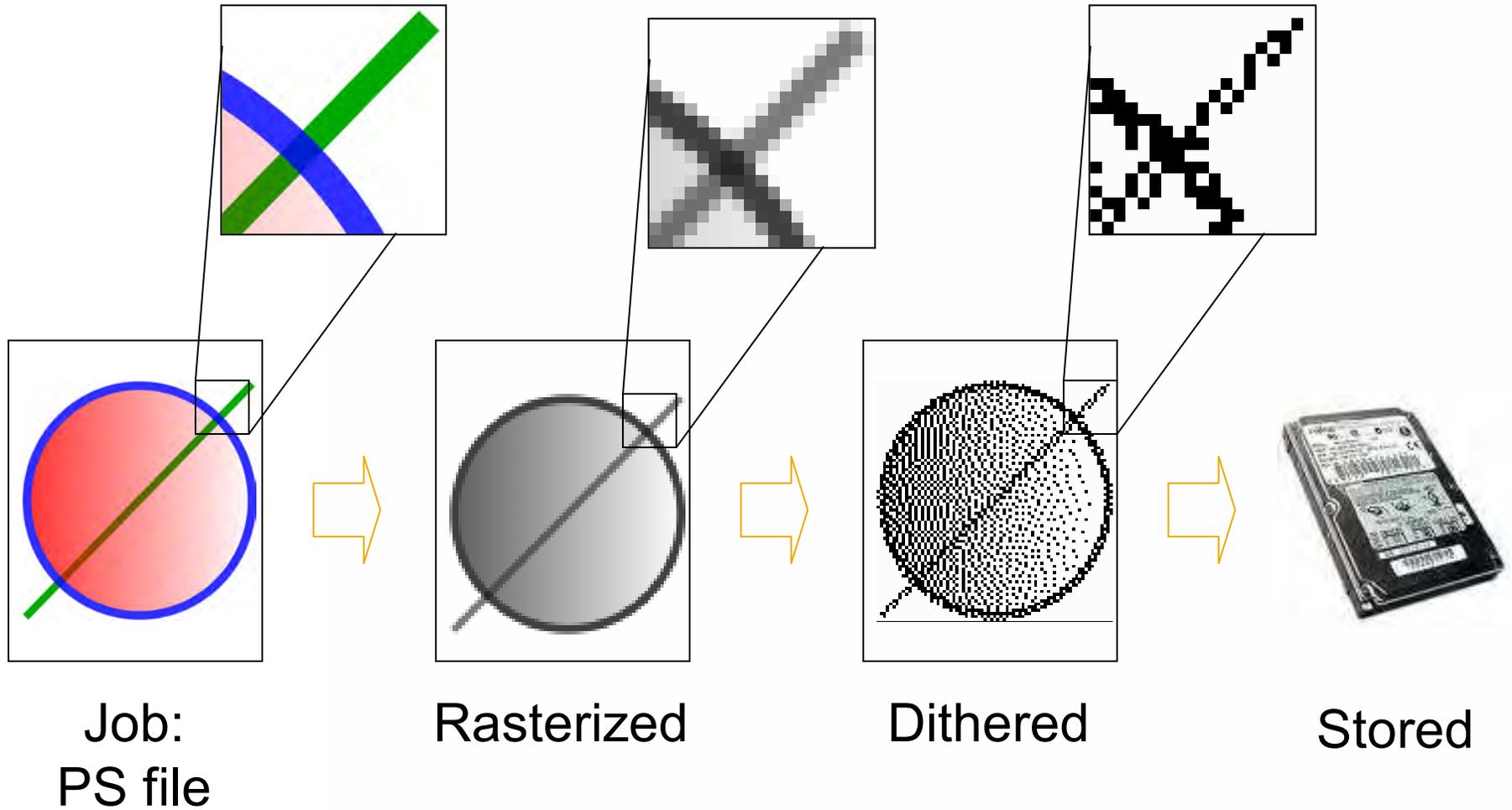


A Laser Printer – black box



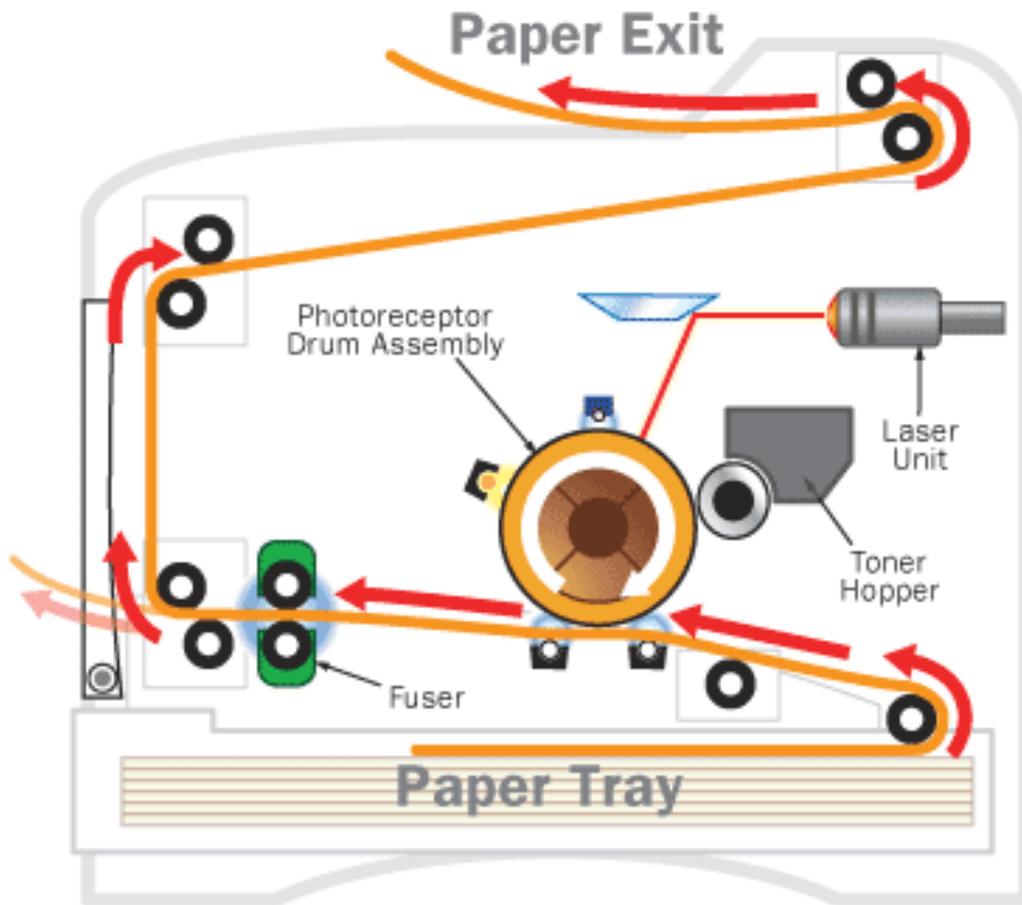


Ripping: from Print Job to Image





The Mechanical Processes: paper



©2005 HowStuffWorks

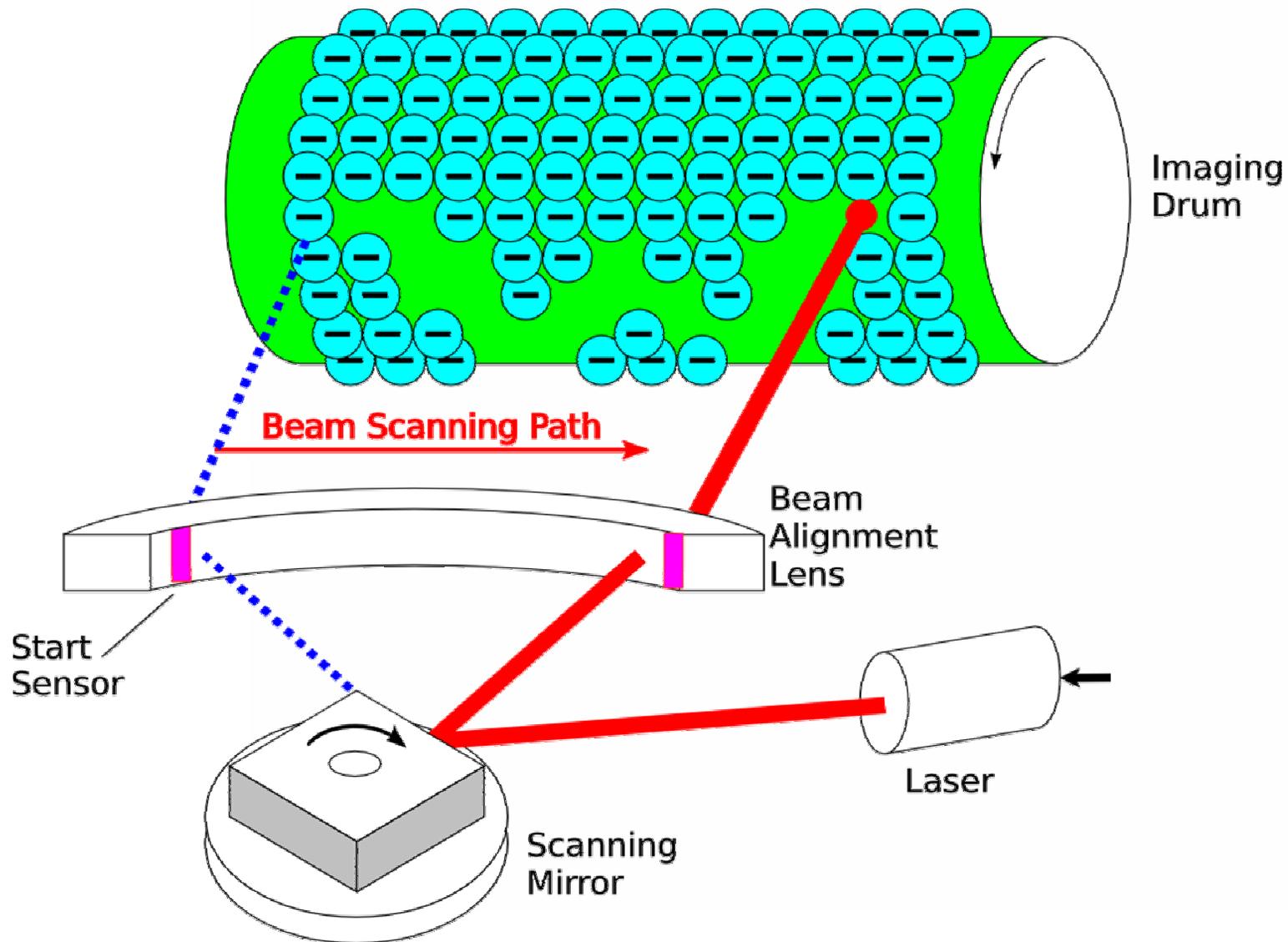
- Several actuators
- Several paper sensors:
 - Tray state
 - Page start/end

Paper *must* run smoothly through the paper path

- Timing depends on it



The Mechanical Processes: print



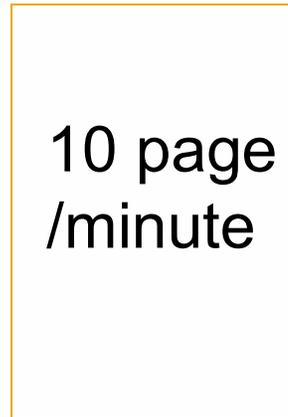


Events and timings

Tray Sensors:

- Tray present
- Paper picked up

8.5" = 5100 dots



11" = 6600 lines

Paper trail:

- Paper ready for drum
- Paper speed

Printing:

- Mirror speed
- Start line sensor
- End line sensor

Paper:

- 10 page/minute
- 120 inch/minute

Laser:

- 10 page/minute
- 70,000 lines/minute
- 5100 dots/line

Various:

- Fuser temperature
- Cancel button
- Network

19.7 millisec / mm

1.2 millisec / line

168 nanosec / dot

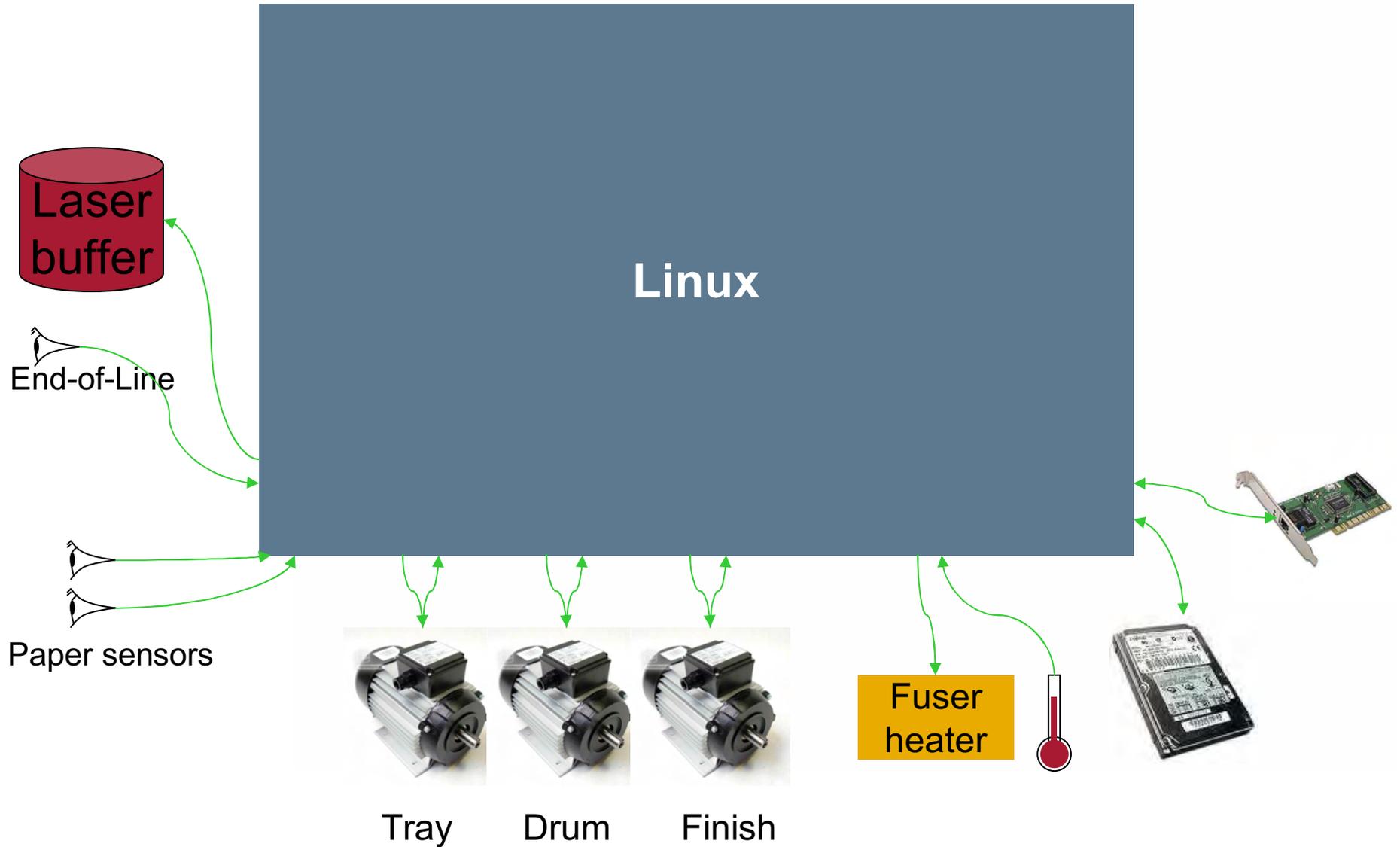


How Real Time?

	1s	Tray present Paper temperature
	100 ms	Cancel button
Linux 2.4 kernel	10 ms	Paper picked up
Linux 2.6 kernel	1 ms	Paper ready for drum Paper speed
	100 us	Load next line buffer
Linux 2.6-rt kernel	10 us	
	1 us	
FG/BG systems	100 ns	Start line + fire
FPGAs	10 ns	Mirror speed
Analog HF electronics	1 ns	
	100 ps	

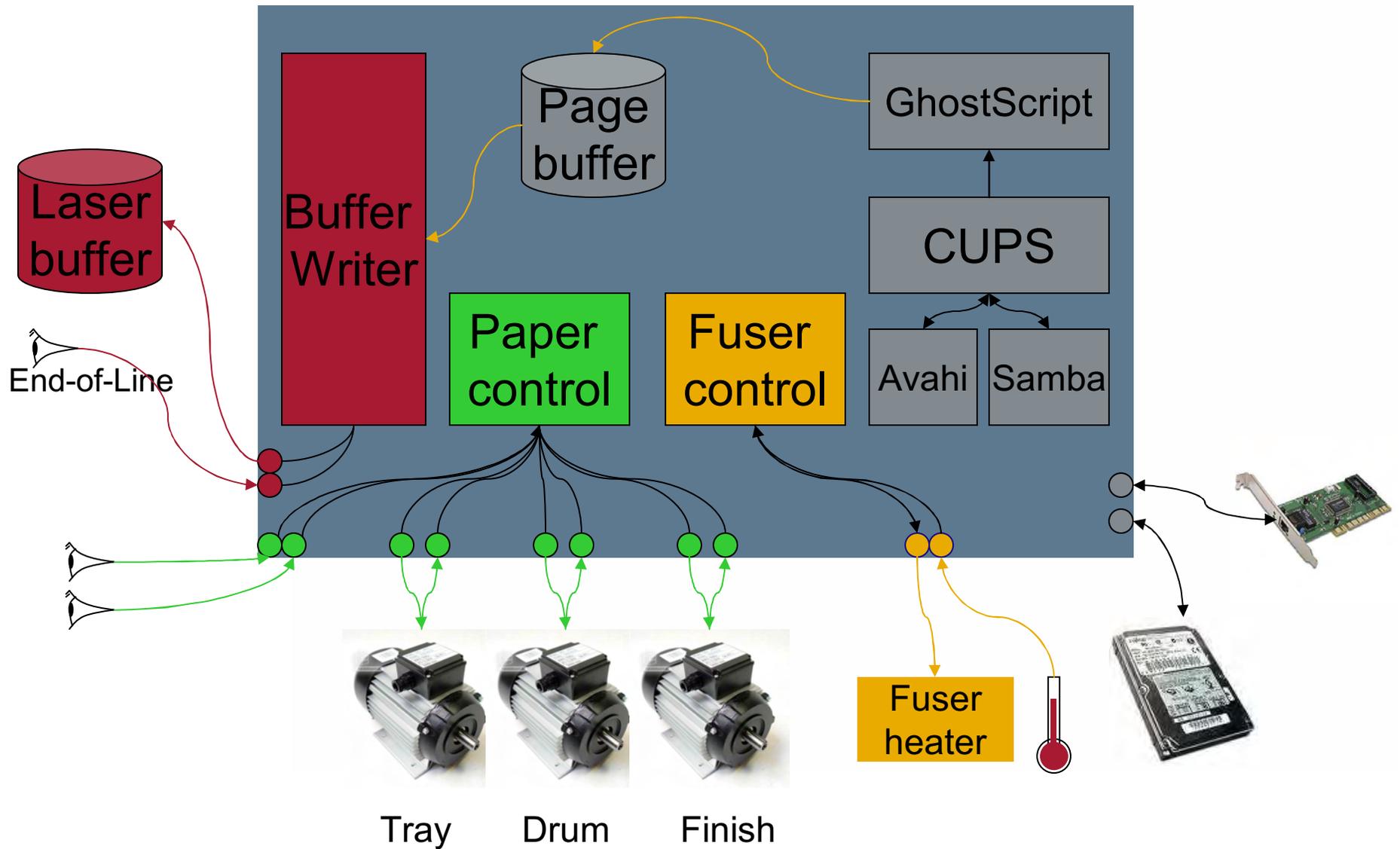


Inputs & Outputs



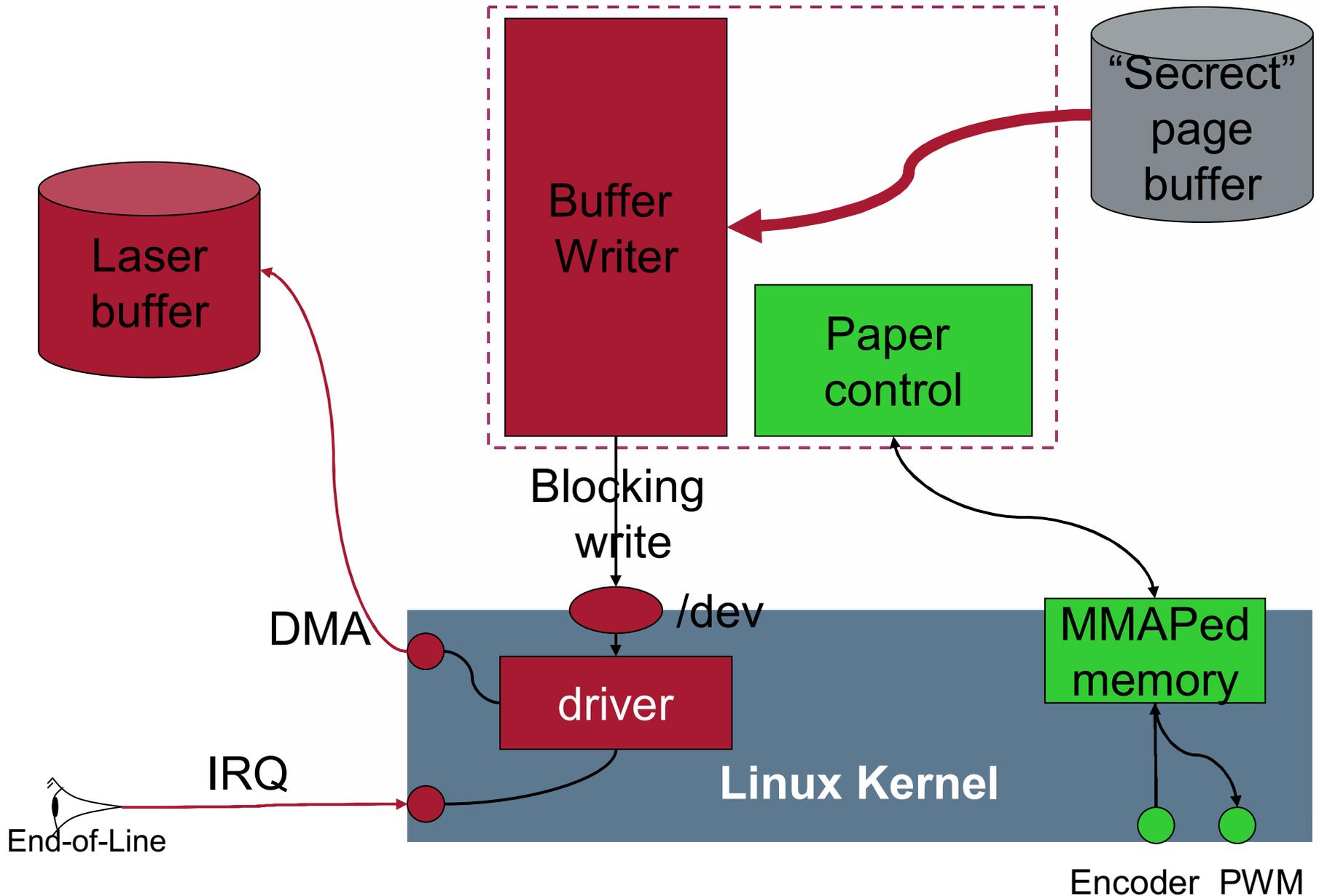


The Block Diagram - details



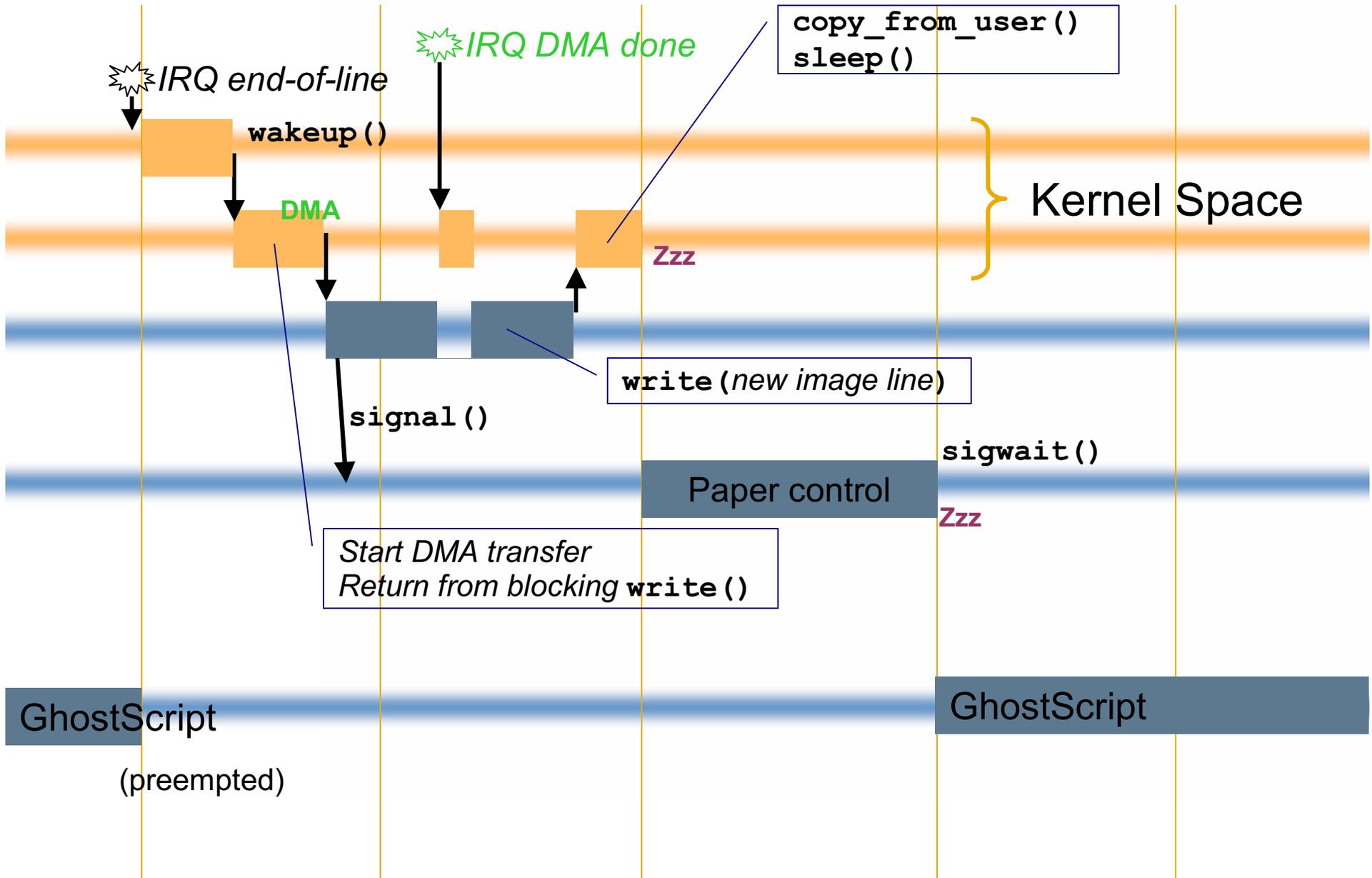


Requirement: User Space Driver





Implementation: data path time line





Implementation: priorities

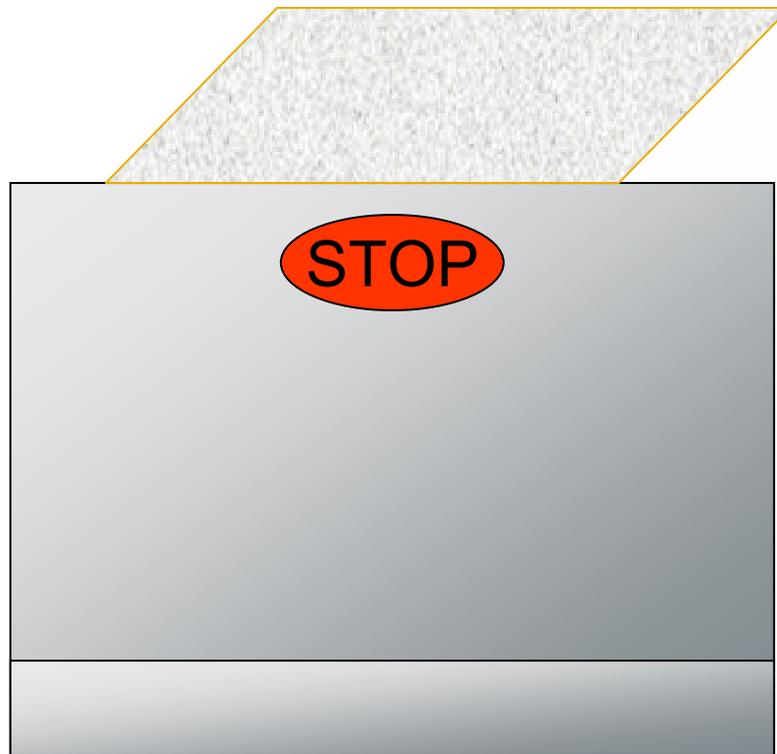
- Buffer driver needs ultimate priority
 - “User space driver” needs 2nd priority
- } These go first
ALWAYS
- Network Queue
 - Harddisk
- } (kernel) I/O bound processes
- Fuser controller
 - But takes very little time
 - Runs of a timer every second?
- CUPS
 - Avahi/Samba
- } Mostly I/O bound processes
- GhostScript RIP
- } CPU/RAM/Disk hog



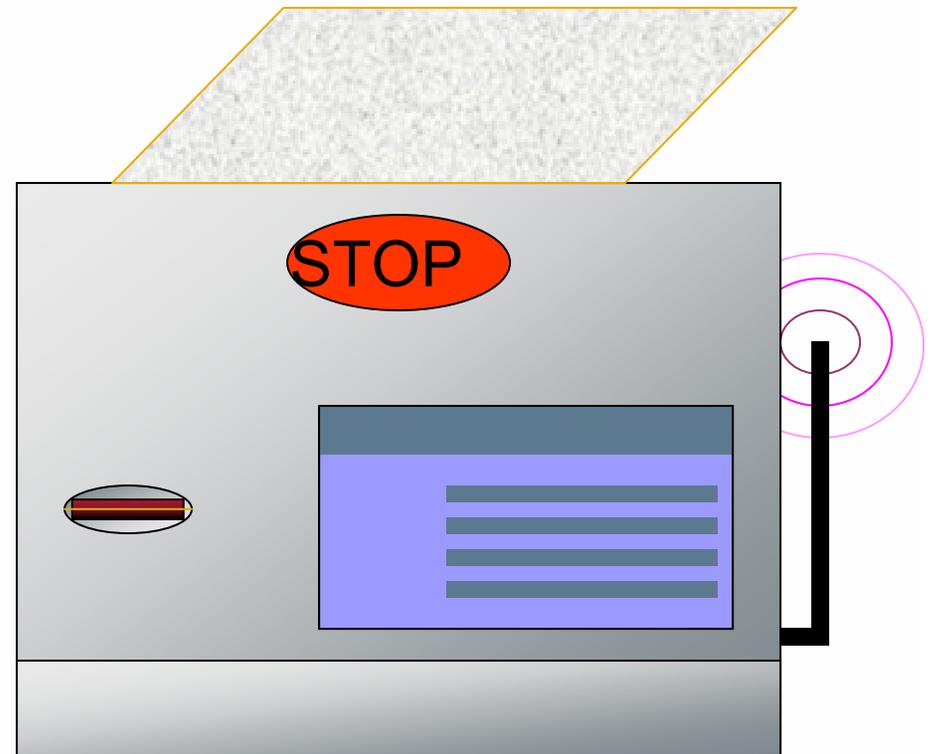
montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Problems to solve

- Many print jobs can be waiting for ripping
 - Store on disk ☺
- GhostScript takes random amounts of RAM
 - enable swap-to-disk to reduce RAM
- Buffer subsystem must always reside in RAM completely
 - full page (4 MB) must be in RAM
 - hard drives are slow (11 ms access time)
 - mlockall (MCL_CURRENT | MCL_FUTURE)
- End-of-Line IRQ should not be a shared IRQ
- Kernel driver is relatively trivial
 - GPL: must publish source to customers



The current printer



With some simple
enhancements



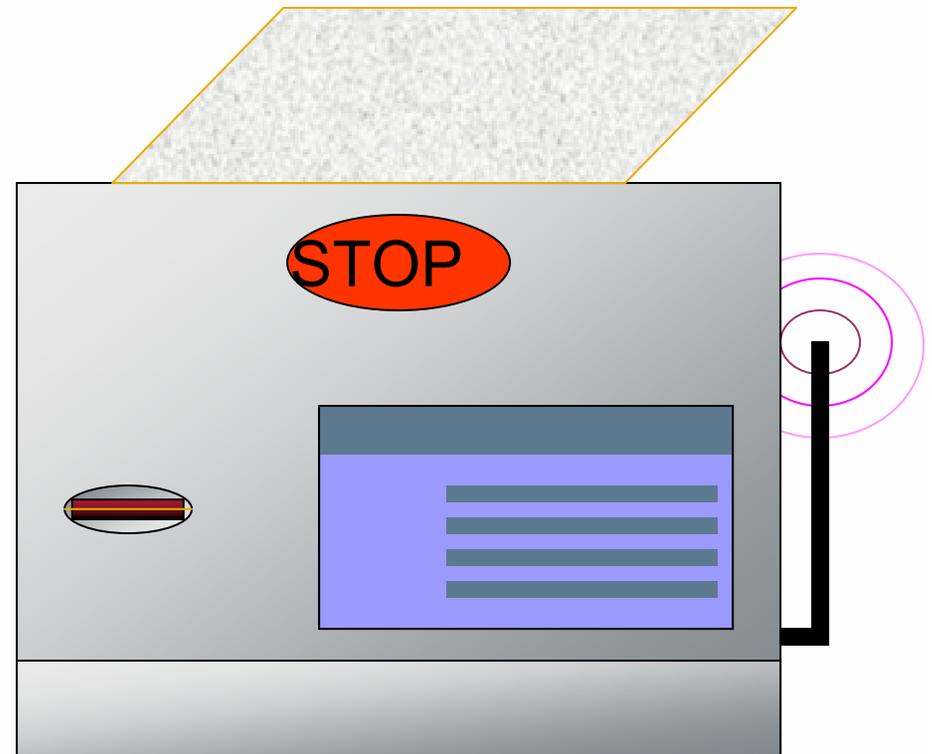
montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

The power of Linux

Marketing



The current printer



With some simple enhancements



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Summary

The RT patch is 3 years old by now

“It will be in the kernel in a year from now”

Several products shipping with it

Still being worked on

Good systems design is hard

Think twice about what to do in software and hardware

RT Patch can solve some problems

Understand your priorities

Draw diagrams

A printer that does 42 pages a minute will attract at least geeks



montavista
VISION
2008
EMBEDDED LINUX DEVELOPERS CONFERENCE

Thank you

Klaas.van.Gend@mvista.com

montavista™