Linux real-time on its way to mainline

Basic lecture
Historical overview about the various steps and components to convert a vanilla Linux kernel into a real-time kernel

Carsten Emde
Open Source Automation Development Lab (OSADL) eG

COOL COMPACT OSADL ONLINE LECTURES

OSADL Open Source Automation Development Lab eG

Agenda

History of making Linux a real-time kernel

- Technology
- Dissemination
- Funding

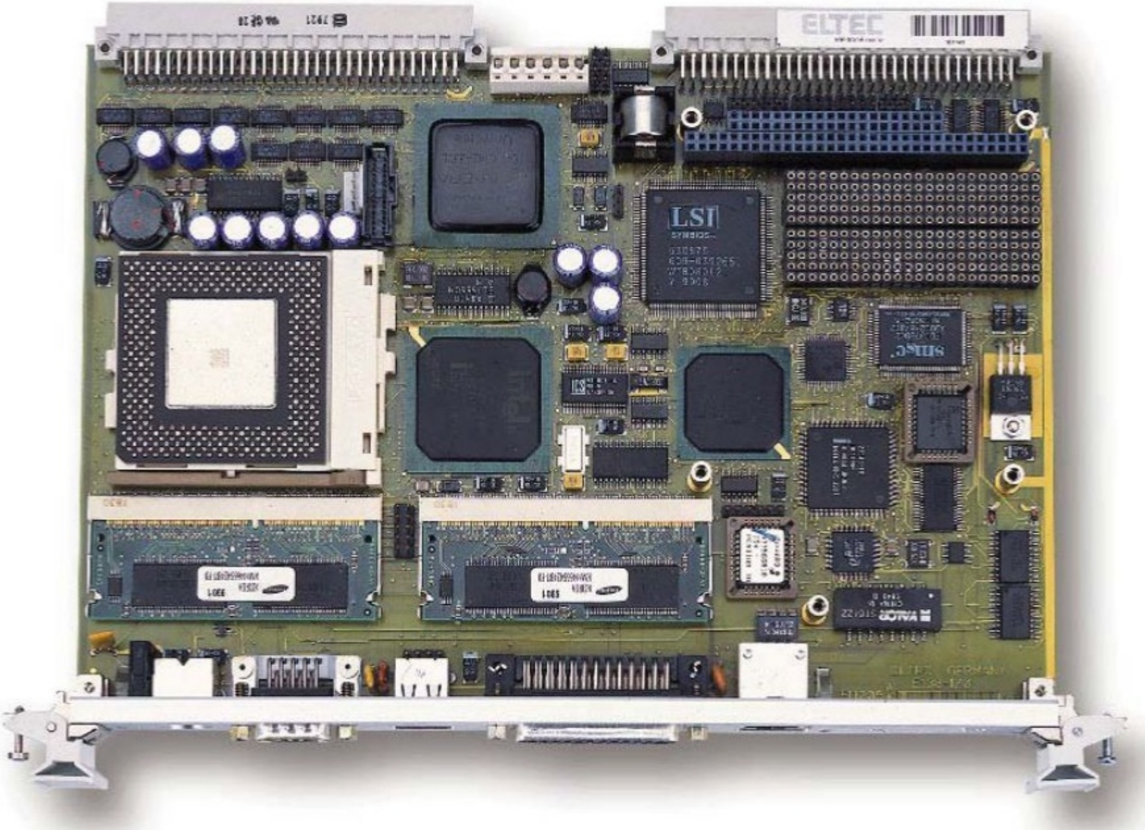
A bit of history

Let's first go back to about 20 years ago.

In the year 2000, most advanced control computers

- were equipped with 32-bit single-core processors
- had Intel x86, Motorola 68k or IBM/Motorola PowerPC architecture
- were running at about 50 to 200 MHz
- had about 32 MByte of memory

As an example an x86-based VMEbus system ...



… that we have under test in our QA Farm even today

Name

rack3slot5.osadl.org

Purpose

Development

Vendor/Board

Eltec/Eurocom-138

Distribution

Debian GNU/Linux 4.0

Kernel

4.18.7-rt5 #15 PREEMPT RT Sun Jan 5 15:12:39 CET 2020 unknown

Uptime

103 days

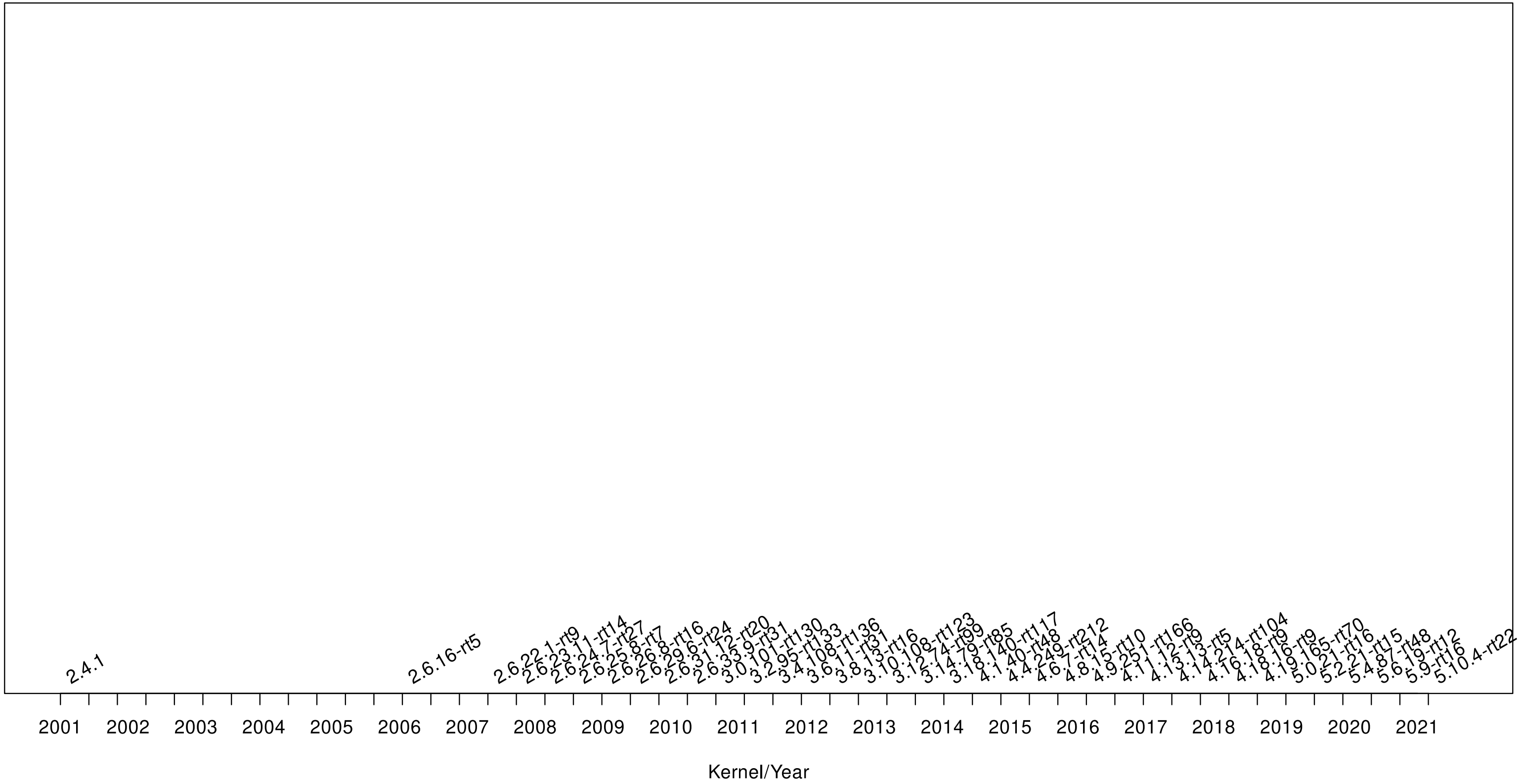
CPU info ⇧

Tag: x86 Intel Pentium @133 MHz

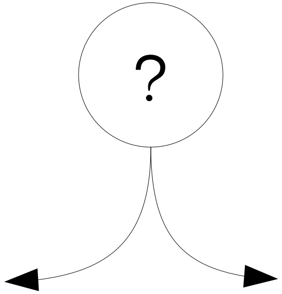
Processor: 0

```
vendor_id       : GenuineIntel
cpu family      : 5
model           : 2
model name      : Pentium 75 - 200
stepping        : 12
cpu MHz         : 132.637
fdiv_bug        : no
f00f_bug        : yes
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr mce cx8 cpuid
bugs            : f00f
bogomips        : 265.27
clflush size    : 32
cache_alignment : 32
address sizes   : 32 bits physical, 32 bits virtual
power management:
```

RT_PREEMPT from 2001 to today



The initial *first-step* fundamental decision



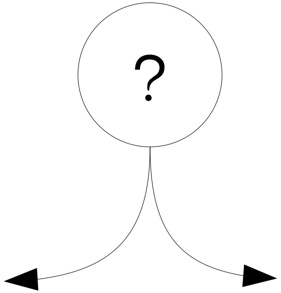
Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision

Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

CD/DVD



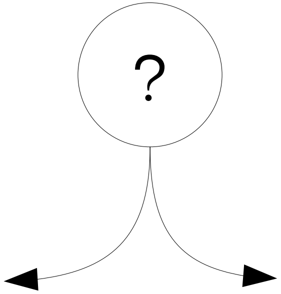
Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision

USB

CD/DVD

Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?



Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision

ISDN

USB

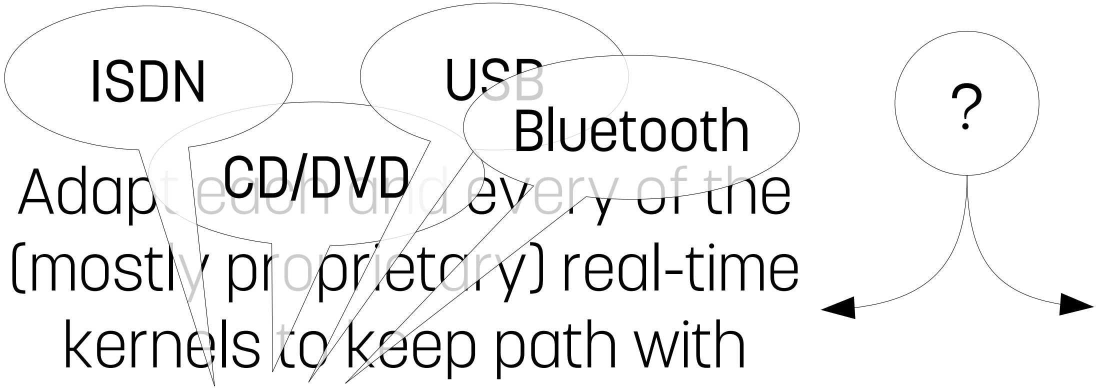
CD/DVD

?

Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision



Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision

ISDN
USB
Bluetooth
CD/DVD
Multicore

?

Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision

ISDN

USB

Bluetooth

CD/DVD

64 Bit

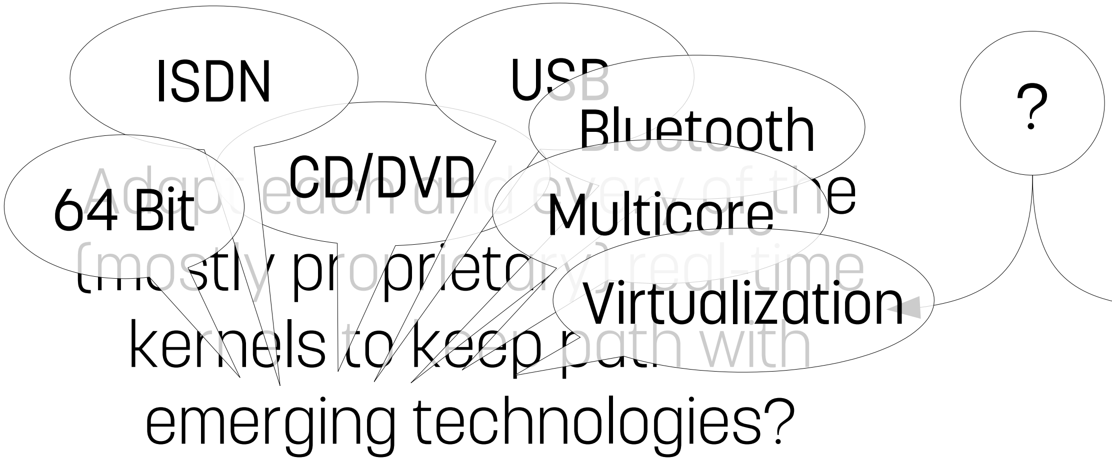
Multicore

Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

?

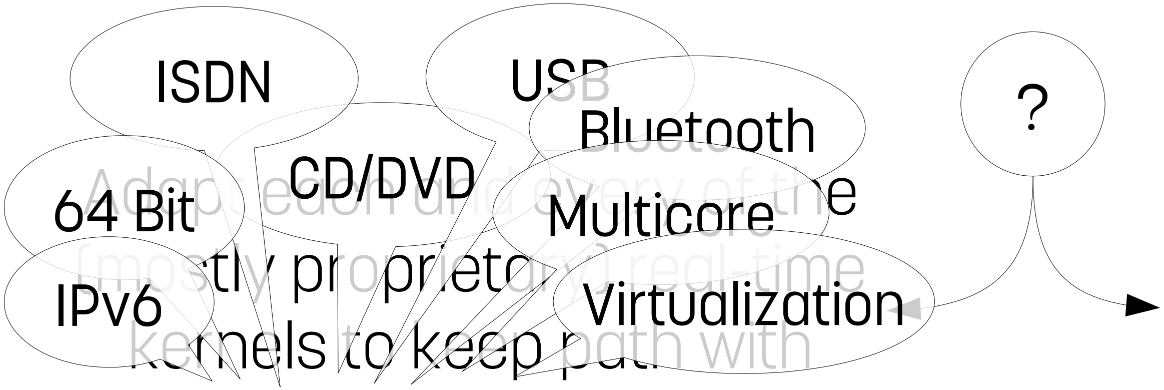
Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision



Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

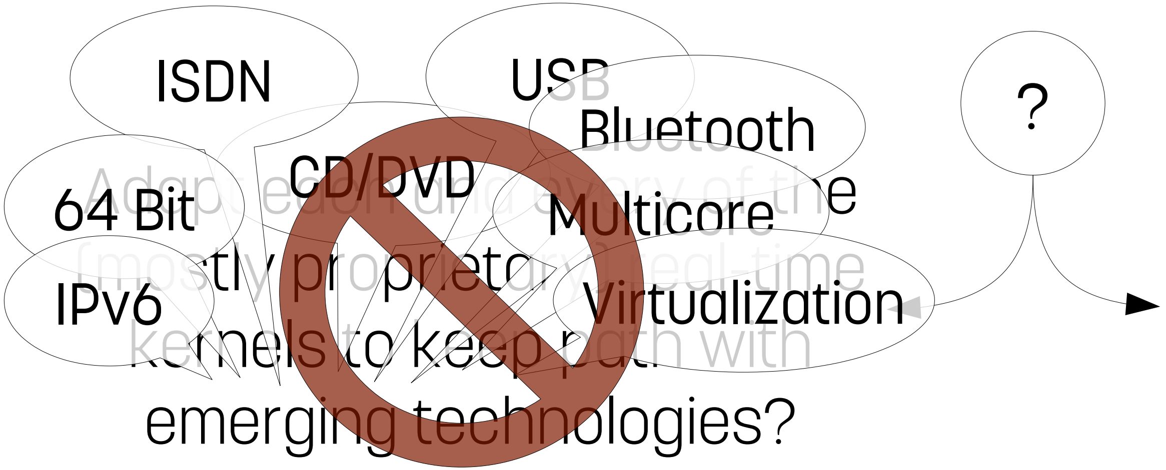
The initial *first-step* fundamental decision



Adapt each and every of the (mostly proprietary) real-time kernels to keep path with emerging technologies?

Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support?

The initial *first-step* fundamental decision



Equip the Linux kernel with generic real-time capabilities and let Linux then provide general real-time support!!!

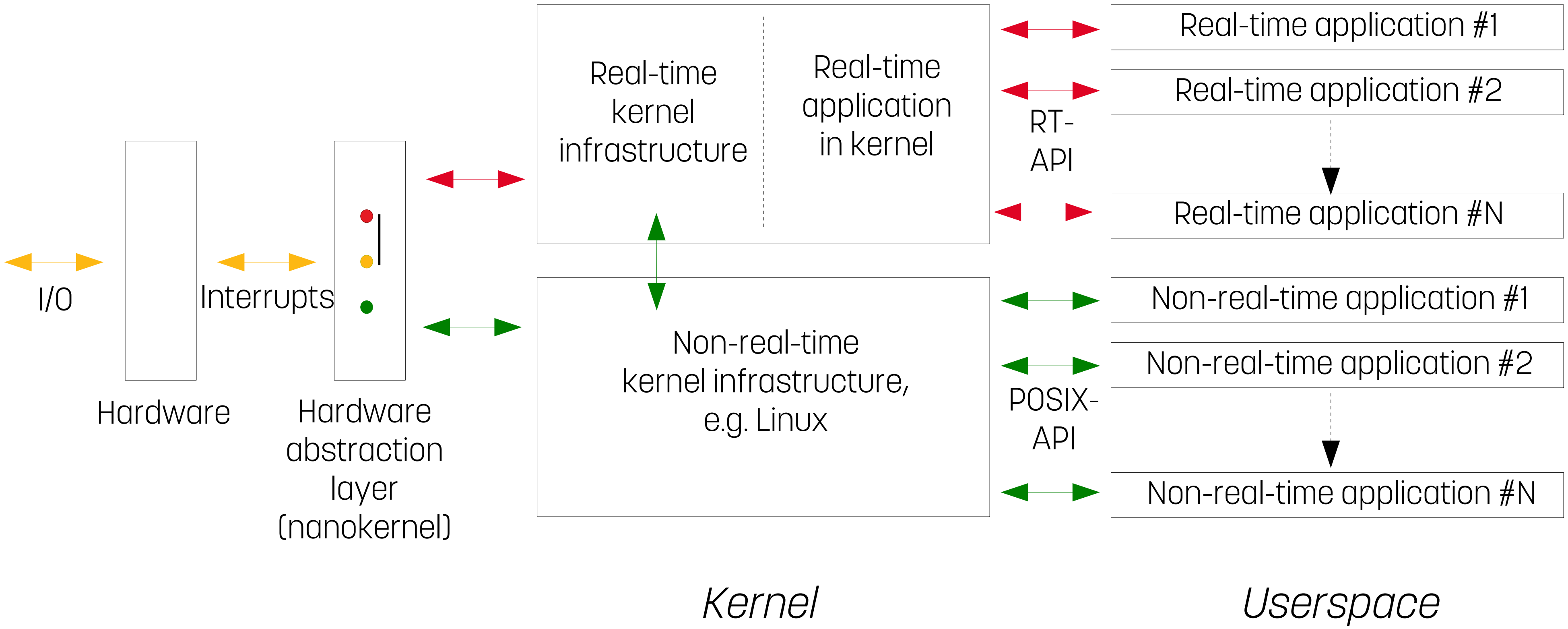
The initial *second-step* fundamental decision

Equip the Linux kernel with
generic real-time capabilities
and let Linux then provide
general real-time support!

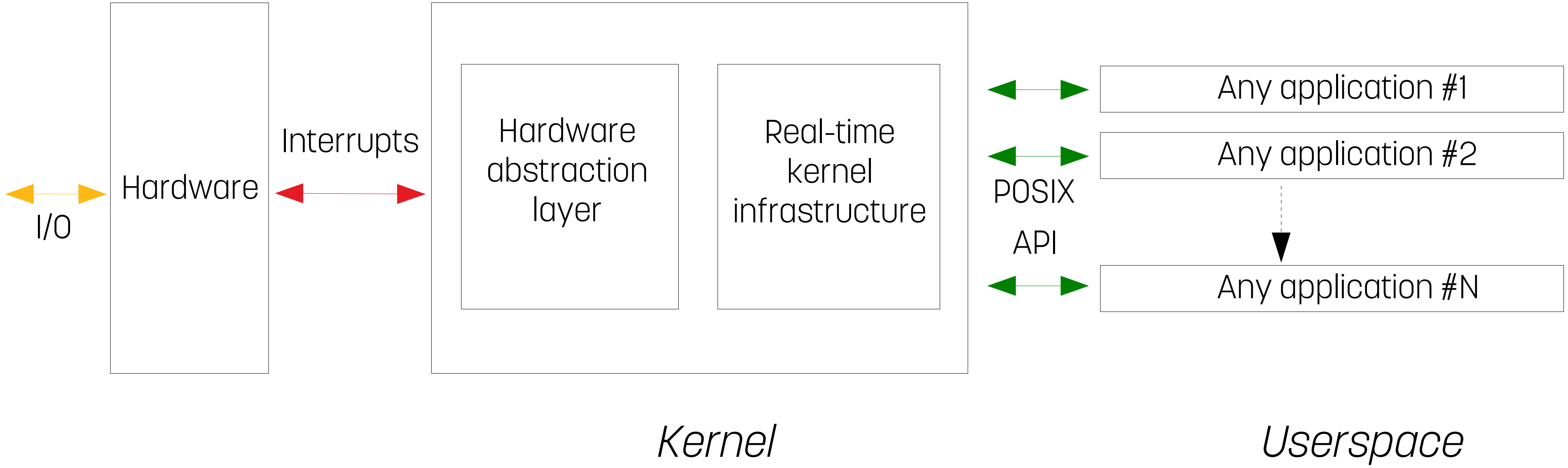
Use a dual-kernel approach
and run Linux in the idle space
of a small RT kernel

Use a single-kernel approach
and convert the Linux kernel
into a fully functional RT kernel

Dual-kernel approach



Single-kernel approach



The majority of real-time specialists ...

... were convinced:

You cannot take a general-purpose operating system kernel that was developed for 10 years and contains about 10 million lines of code and simply retrofit it to become a real-time kernel.

A researcher's opinion

Current Research Efforts in
Real-time and Embedded Systems

Douglas Niehaus
Information and Telecommunication Technology Center
Electrical Engineering and Computer Science Department
University of Kansas
niehaus@ittc.ku.edu

Information and Telecommunication Technology Center — University of Kansas

A researcher's opinion

Real-Time and Embedded Systems

- Change many of the assumptions underlying conventional computer system design
- Real-time requires finer resolution time keeping and resource allocation because they must control *when* actions occur
  - Precise control of events on real-time line
  - When a computation executes is part of its correctness
  - Execution time predictions are required in many cases
- Embedded systems are often special purpose
  - Application semantics differ widely
  - Specialized & Restricted semantics → specialized programming models
  - No single programming model is best match for all application semantics → multiple models or lowest common denominator
- Majority of all computers (80%+) are embedded, increasing number must satisfy real-time constraints

Information and Telecommunication Technology Center

University of Kansas

A researcher's opinion

Real-Time and Embedded Systems

- Change many of the assumptions underlying conventional computer system design
- Real-time requires finer resolution time keeping and resource allocation because they must control *when* actions occur
  - Precise control of events on real-time line
  - When a computation executes is part of its correctness
  - Execution time predictions are required in many cases
- Embedded systems are often special purpose
  - Application semantics differ widely
  - Specialized & Restricted semantics → specialized programming models
  - No single programming model is best match for all application semantics → multiple models or lowest common denominator
- Majority of all computers (80%+) are embedded, increasing number must satisfy real-time constraints

Information and Telecommunication Technology Center

University of Kansas

A researcher's opinion

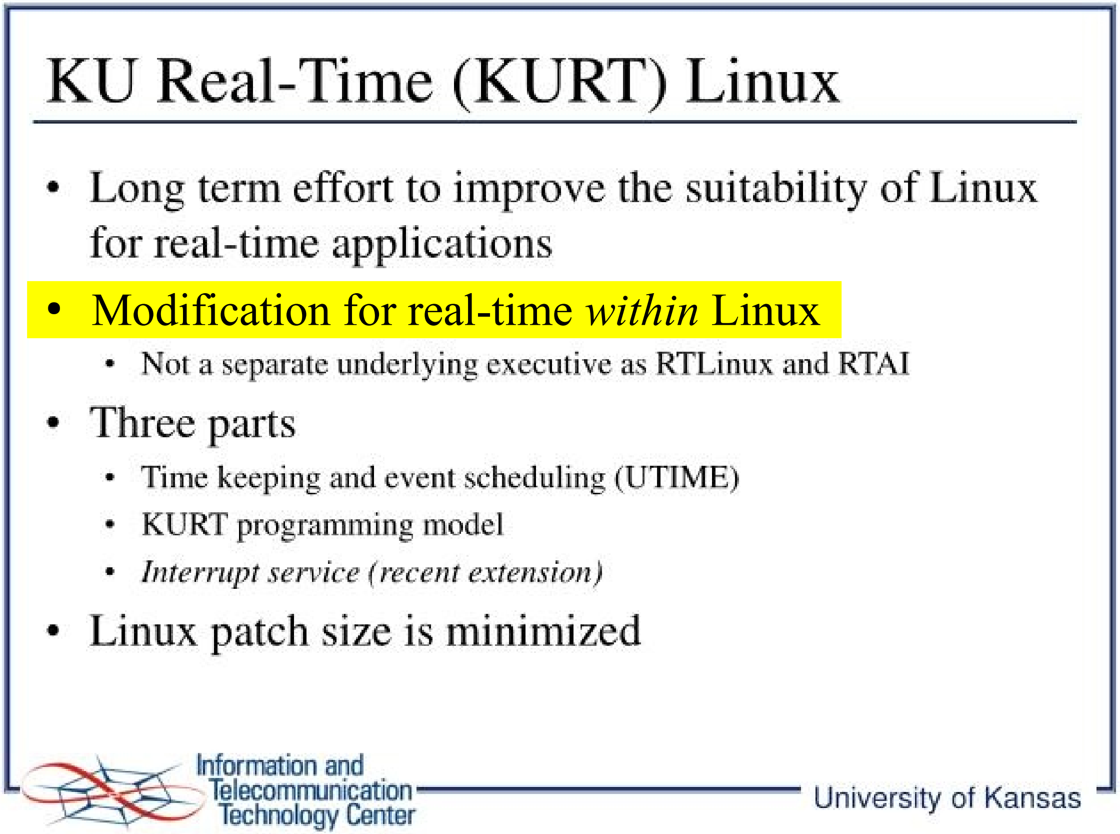
KU Real-Time (KURT) Linux

- Long term effort to improve the suitability of Linux for real-time applications
- Modification for real-time *within* Linux
  - Not a separate underlying executive as RTLinux and RTAI
- Three parts
  - Time keeping and event scheduling (UTIME)
  - KURT programming model
  - *Interrupt service (recent extension)*
- Linux patch size is minimized

Information and Telecommunication Technology Center

University of Kansas

A researcher's opinion



The initial *second-step* fundamental decision

Equip the Linux kernel with
generic real-time capabilities
and let Linux then provide
general real-time support!

Use a two-kernel approach and
run Linux in the idle space of a
small RT kernel

Use a single-kernel approach
and convert the Linux kernel
into a fully functional RT kernel

The initial *second-step* fundamental decision

Equip the Linux kernel with
generic real-time capabilities
and let Linux then provide
general real-time support!

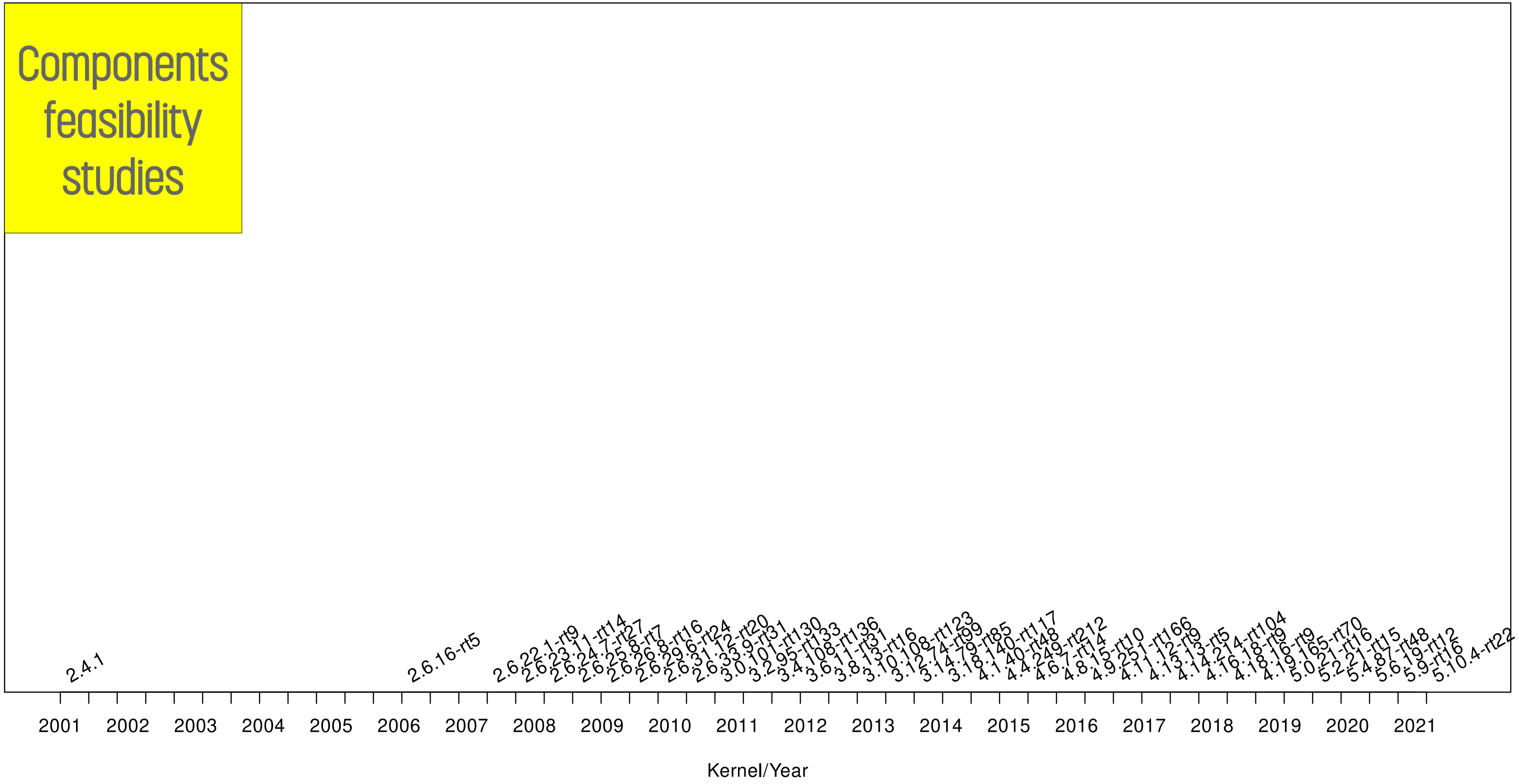
Use a two-kernel approach and run Linux in the idle space of a small RT kernel

RTAI, Xenomai, RTCore

Use a single-kernel approach and convert the Linux kernel into a fully functional RT kernel

RT_PREEMPT

RT_PREEMPT from 2001 to today: Technology



A researcher's proposal

UTime: Time Keeping & Event Scheduling

- Portable High Resolution API
  - **Time Standard**: Pentium Time Stamp Counter
    - CPU clock (nanosecond) resolution
  - **Next Event Interrupt**: microsecond resolution
    - PC timer Chip (8159) or Pentium PIC
- Useful in its own right
  - Often used without KURT component
  - Starting point of Linux High Resolution Timers Project
- Multiple Platforms
  - StrongARM, XScale/FPGA SBC, AMD Elan (x86+), Power PC (PPC) Virtex II Pro SBC (future)
  - *Time Standard* and *Next Event Interrupt* methods vary

Information and Telecommunication Technology Center

University of Kansas

A researcher's proposal

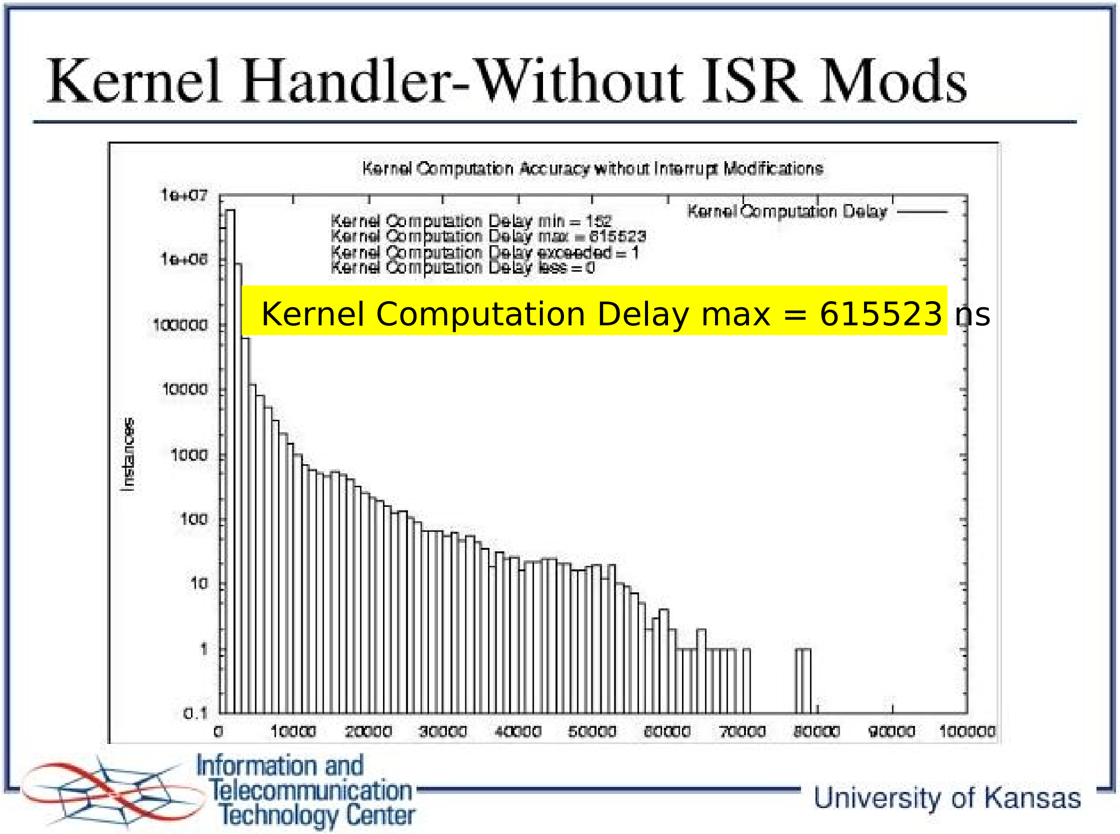
UTime: Time Keeping & Event Scheduling

- Portable High Resolution API
  - **Time Standard**: Pentium Time Stamp Counter
    - CPU clock (nanosecond) resolution
  - **Next Event Interrupt**: microsecond resolution
    - PC timer Chip (8159) or Pentium PIC
- Useful in its own right
  - Often used without KURT component
  - Starting point of Linux High Resolution Timers Project
- Multiple Platforms
  - StrongARM, XScale/FPGA SBC, AMD Elan (x86+), Power PC (PPC) Virtex II Pro SBC (future)
  - *Time Standard* and *Next Event Interrupt* methods vary

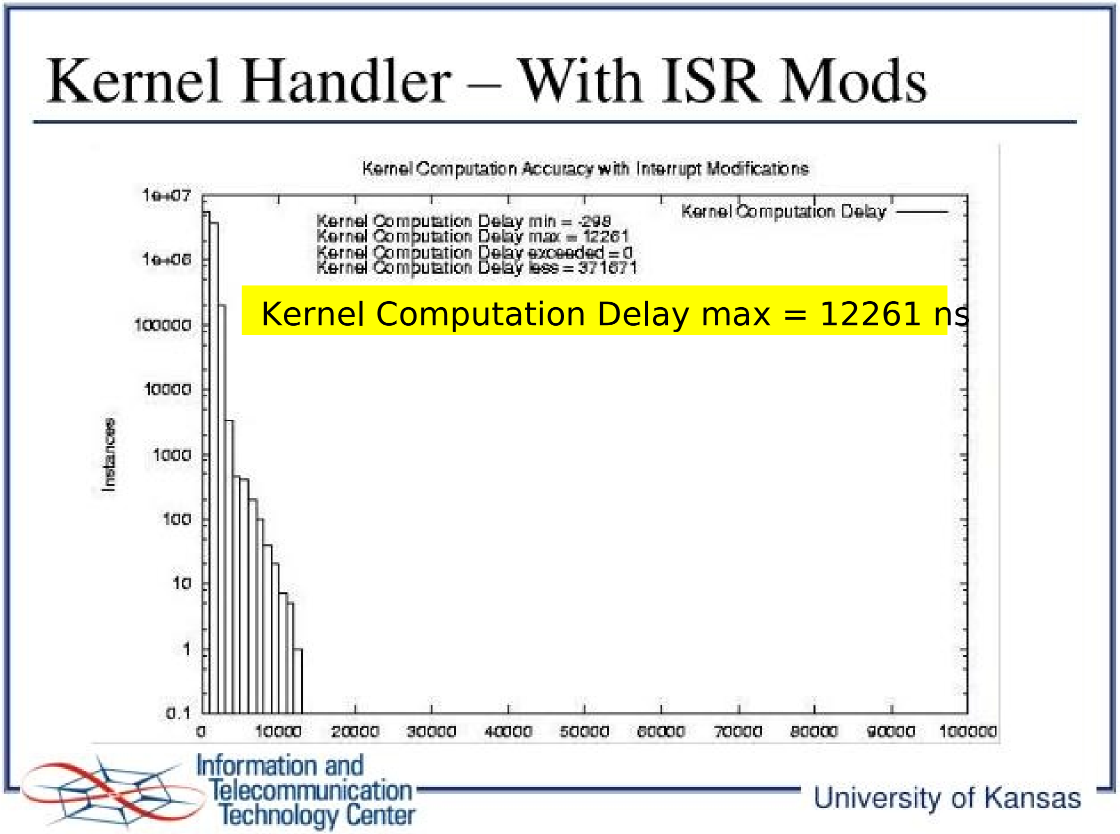
Information and Telecommunication Technology Center

University of Kansas

A researcher's proposal



A researcher's proposal

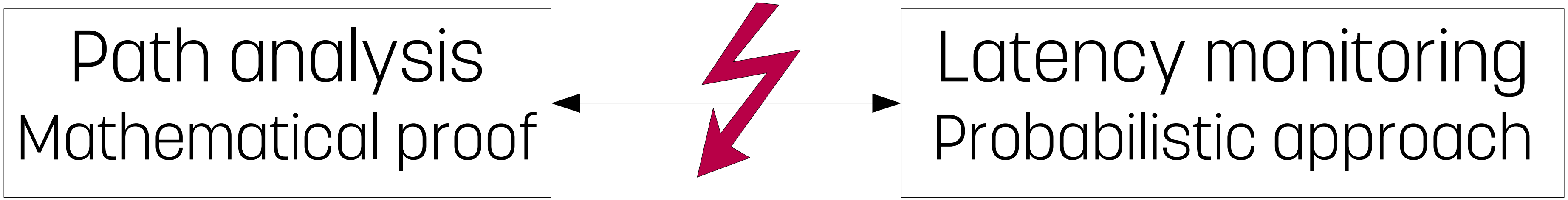


But then the majority of real-time specialists …

… came up with another hurdle:

Even if you believe you could create a real-time kernel with 10 million lines of code, you can never prove that it is real-time, since the silver bullet of real-time proof, the path analysis, does not work.

And so the path-analysis war broke out



What is path analysis?

```
i = dram[0];
i++;
dram[0] = i;
```

```
movea.l   #dram,a0
move.l    (a0),d0
add.l     #1,d0
move.l    d0,(a0)
```

Motorola MC68000
@ 8 MHz
500 Dhrystones/s

```
mov     dram,eax
mov     eax,-4(ebp)
addl    $1,-4(ebp)
mov     -4(ebp),eax
mov     eax,dram
```

Intel x86
Skylake 10x2-core @ 4 GHz
500.000.000 Dhrystones/s

Look up execution time

7.2 MOVE INSTRUCTION EXECUTION TIMES

Tables 7-2, 7-3, and 7-4 list the numbers of clock periods for the move instructions. The totals include instruction fetch, operand reads, and operand writes. The total number of clock periods, the number of read cycles, and the number of write cycles are shown in the previously described format.

Table 7-2. Move Byte Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	–(An)	(d_{16}, An)	(d_8, An, Xn)*	(xxx).W	(xxx).L
Dn	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	20(4/1)	22(4/1)	20(4/1)	28(6/1)
An	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	20(4/1)	22(4/1)	20(4/1)	28(6/1)
(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	24(5/1)	26(5/1)	24(5/1)	32(7/1)
(An)+	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	24(5/1)	26(5/1)	24(5/1)	32(7/1)
–(An)	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	26(5/1)	28(5/1)	26(5/1)	34(7/1)
(d_{16}, An)	20(5/0)	20(5/0)	24(5/1)	24(5/1)	24(5/1)	32(7/1)	34(7/1)	32(7/1)	40(9/1)
(d_8, An, Xn)*	22(5/0)	22(5/0)	26(5/1)	26(5/1)	26(5/1)	34(7/1)	36(7/1)	34(7/1)	42(9/1)
(xxx).W	20(5/0)	20(5/0)	24(5/1)	24(5/1)	24(5/1)	32(7/1)	34(7/1)	32(7/1)	40(9/1)
(xxx).L	28(7/0)	28(7/0)	32(7/1)	32(7/1)	32(7/1)	40(9/1)	42(9/1)	40(9/1)	48(11/1)
(d_{16}, PC)	20(5/0)	20(5/0)	24(5/1)	24(5/1)	24(5/1)	32(7/1)	34(7/1)	32(7/1)	40(9/1)
(d_8, PC, Xn)*	22(5/0)	22(5/0)	26(5/1)	26(5/1)	26(5/1)	34(7/1)	36(7/1)	34(7/1)	42(9/1)
#<data>	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	28(6/1)	30(6/1)	28(6/1)	36(8/1)

*The size of the index register (Xn) does not affect execution time.

Use path analysis

```
i = dram[0];
i++;
dram[0] = i;
```

```
movea.l   #dram,a0
move.l    (a0),d0
add.l     #1,d0
move.l    d0,(a0)
```

Load instructions from memory to register and execute them.
Duration: **56** processor cycles

```
mov     dram,eax
mov     eax,-4(ebp)
addl    $1,-4(ebp)
mov     -4(ebp),eax
mov     eax,dram
```

Motorola MC68000
@ 8 MHz
500 Dhrystones/s

Intel x86
Skylake 10x2-core @ 4 GHz
500.000.000 Dhrystones/s

Path analysis in modern processors ...

```
i = dram[0];
i++;
dram[0] = i;
```

```
movea.l   #dram,a0
move.l    (a0),d0
add.l     #1,d0
move.l    d0,(a0)
```

Load instructions from memory to register and execute them.
Duration: **??** processor cycles

```
mov    dram,eax
mov    eax,-4(ebp)
addl   $1,-4(ebp)
mov    -4(ebp),eax
mov    eax,dram
```

Motorola MC68000
@ 8 MHz
500 Dhrystones/s

Intel x86
Skylake 10x2-core @ 4 GHz
500.000.000 Dhrystones/s

Path analysis in modern processors ... no longer works

```
i = dram[0];
i++;
dram[0] = i;
```

```
movea.l   #dram,a0
move.l    (a0),d0
add.l     #1,d0
move.l    d0,(a0)
```

Motorola MC68000
@ 8 MHz
500 Dhrystones/s

Load instructions from memory to register and execute them.
Duration: **??** processor cycles

```
mov    dram,eax
mov    eax,-4(ebp)
addl   $1,-4(ebp)
mov    -4(ebp),eax
mov    eax,dram
```

Intel x86
Skylake 10x2-core @ 4 GHz
500.000.000 Dhrystones/s

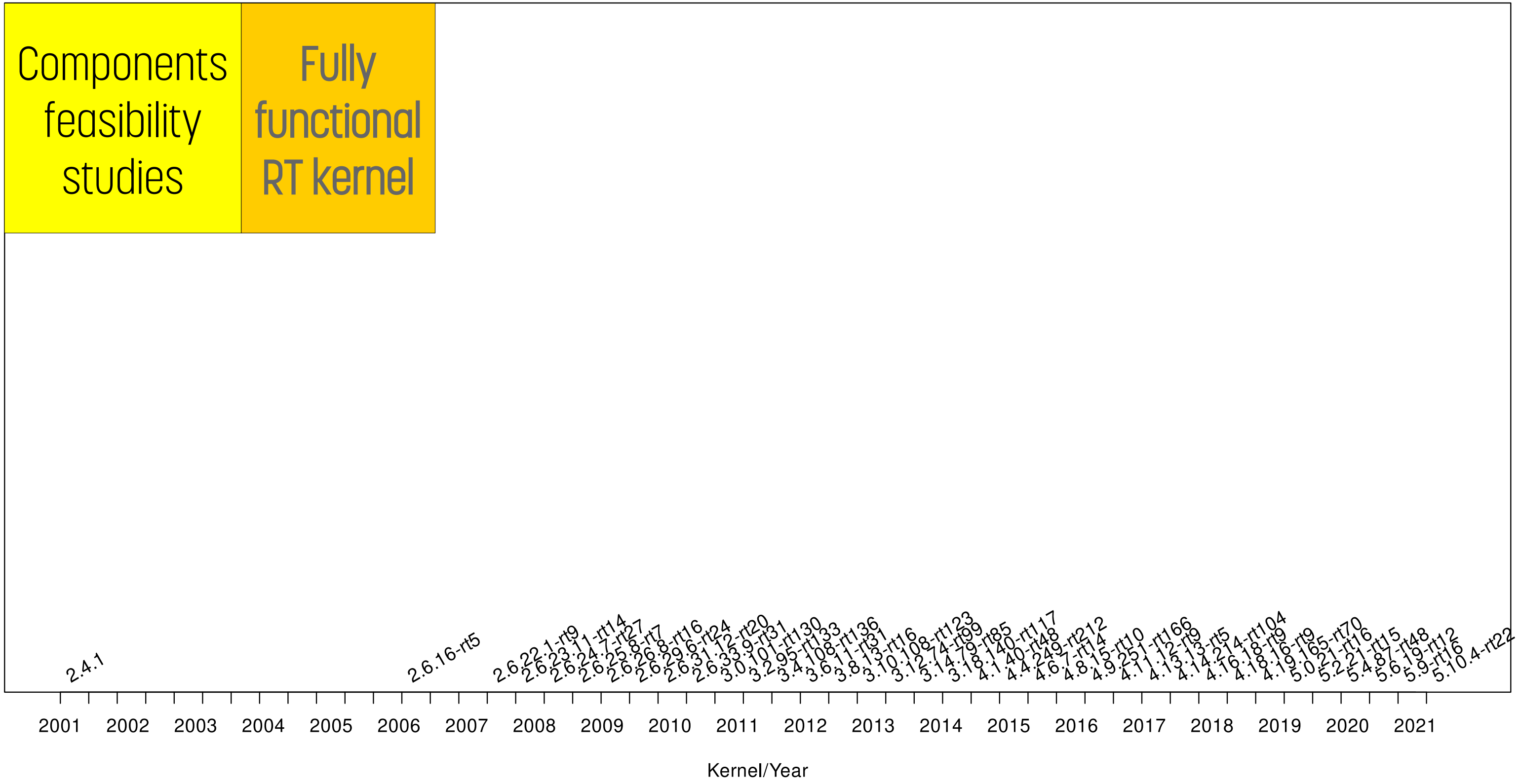
Instruction not in cache/no free cache lines

Data not in cache/no free cache lines

System Management Interrupt

Instruction may be emulated (microcode patch)

RT_PREEMPT from 2001 to today: Technology



PREEMPT_RT gains momentum

LWN
.net
News from the source

User: Password: Log in | Subscribe | Register

Content
Weekly Edition
Archives
Search
Kernel
Security
Distributions
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Kernel page

2.6.16-rt5

From: Ingo Molnar <mingo@elte.hu>
To: linux-kernel@vger.kernel.org
Subject: 2.6.16-rt5
Date: Thu, 23 Mar 2006 09:17:08 +0100
Cc: Thomas Gleixner <tglx@linutronix.de>

```
i have released the 2.6.16-rt5 tree, which can be downloaded from the
usual place:

   http://redhat.com/~mingo/realtime-preempt/

there's been quite some churn since -rt1:

 244 files changed, 1806 insertions(+), 1588 deletions(-)

this was mostly due to the simplification of IRQ-flag handling:
local_irq_*() now defaults to using the raw IRQ flags. The 'soft
irq-flag' code only had a debugging purpose, but that purpose is equally
well suited by dont-schedule-while-in-atomic-section checks. This
cleanup resulted in a nice 10% reduction of the -rt patch's size, and
should make porting to architectures simpler.

another bigger change is the continued rework of the PI code by Thomas
Gleixner: it should now be Bug Free (tm) - in particular the SMP locking
deadlock noticed by Esben Nielsen should be fixed. There are also lots
of updates to the PI-futex code too, by Thomas.

there are also lots of smaller fixes for regressions in -rt1.

to build a 2.6.16-rt5 tree, the following patches should be applied:

  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.16.tar.bz2
  http://redhat.com/~mingo/realtime-preempt/patch-2.6.16-rt5

        Ingo
-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
Please read the FAQ at  http://www.tux.org/lkml/
```

PREEMPT_RT gains momentum

LWN.net
News from the source

User: Password: Log in | Subscribe | Register

2.6.16-rt5

Content
Weekly Edition
Archives
Search
Kernel
Security
Distributions
Events calendar
Unread comments

LWN FAQ
Write for us

Edition
Return to the Kernel page

From:	Ingo Molnar <mingo@elte.hu>
To:	linux-kernel@vger.kernel.org
Subject:	2.6.16-rt
Date:	Thu, 23 Mar 2006 09:17:08+0100
CC:	Thomas Gleixner <tglx@linutronix.de>

```
i have released the 2.6.16-rt5 tree, which can be downloaded from the
usual place:

   http://redhat.com/~mingo/realtime-preempt/

there's been quite some churn since -rt1:

 244 files changed, 1806 insertions(+), 1588 deletions(-)

this was mostly due to the simplification of IRQ-flag handling:
local_irq_*() now defaults to using the raw IRQ flags. The 'soft
irq-flag' code only had a debugging purpose, but that purpose is equally
well suited by dont-schedule-while-in-atomic-section checks. This
cleanup resulted in a nice 10% reduction of the -rt patch's size, and
should make porting to architectures simpler.

another bigger change is the continued rework of the PI code by Thomas
Gleixner: it should now be Bug Free (tm) - in particular the SMP locking
deadlock noticed by Esben Nielsen should be fixed. There are also lots
of updates to the PI-futex code too, by Thomas.

there are also lots of smaller fixes for regressions in -rt1.

to build a 2.6.16-rt5 tree, the following patches should be applied:

  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.16.tar.bz2
  http://redhat.com/~mingo/realtime-preempt/patch-2.6.16-rt5

        Ingo
-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
Please read the FAQ at  http://www.tux.org/lkml/
```

PREEMPT_RT gains momentum

LWN.net
News from the source

User: Password: Log in | Subscribe | Register

2.6.16-rt5

From:	Ingo Molnar <mingo@elte.hu>
To:	linux-kernel@vger.kernel.org
Subject:	2.6.16-rt
Date:	Thu, 23 Mar 2006 09:17:08+0100
CC:	Thomas Gleixner <tglx@linutronix.de>

LWN FAQ
Write for us

Edition
Return to the Kernel page

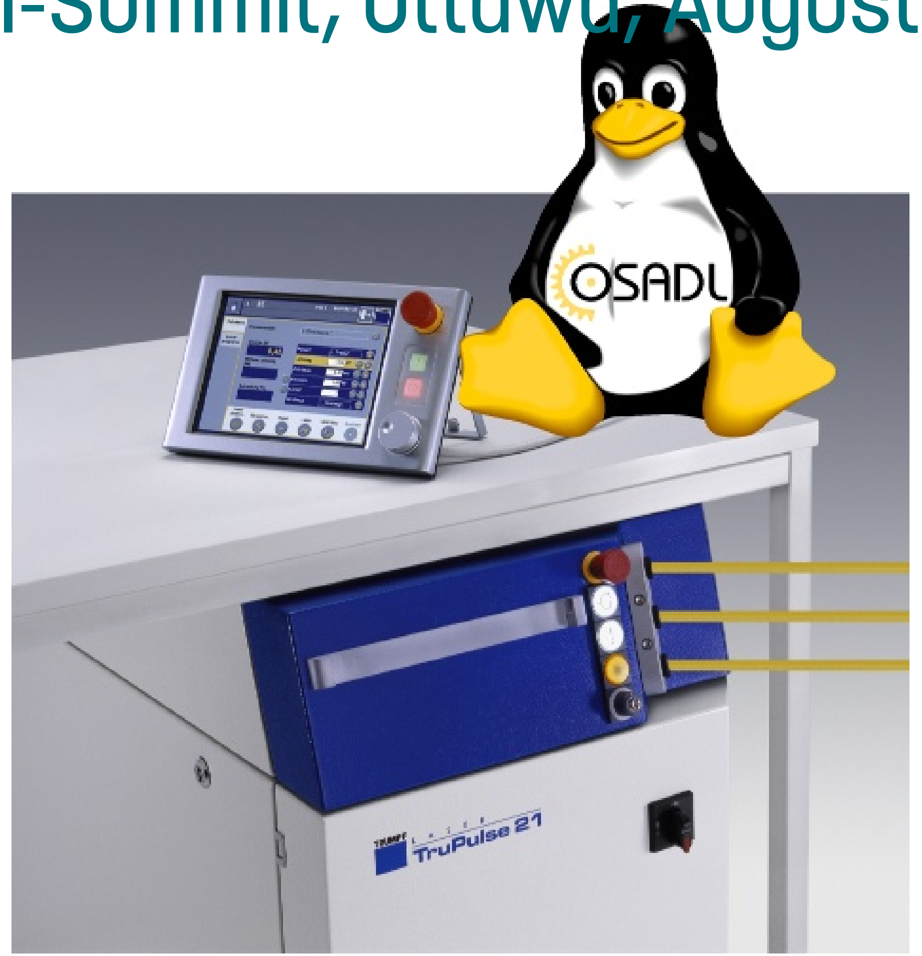
```
 244 files changed, 1806 insertions(+), 1588 deletions(-)

this was mostly due to the simplification of IRQ-flag handling:
local_irq_*() now defaults to using the raw IRQ flags. The 'soft
irq-flag' code only had a debugging purpose, but that purpose is equally
```

another bigger change is the continued rework of the PI code by Thomas Gleixner: it should now be Bug Free (tm) - in particular the SMP locking deadlock noticed by Esben Nielsen should be fixed. There are also lots of updates to the PI-futex code too, by Thomas.

```
        Ingo
-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
Please read the FAQ at  http://www.tux.org/lkml/
```

Kernel-Summit, Ottawa, August 2006

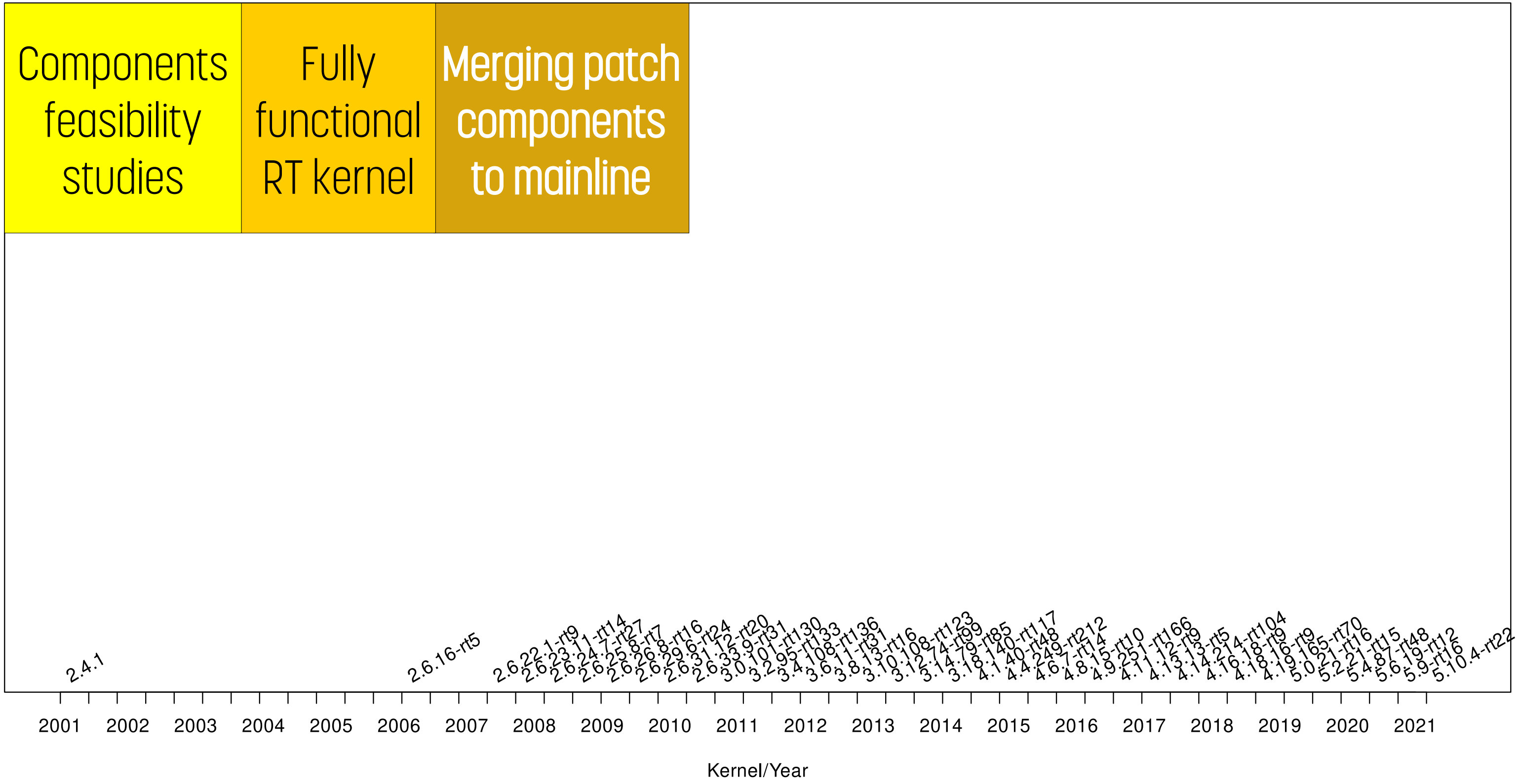


Kernel-Summit, Ottawa, August 2006

"Controlling a laser with Linux is crazy, but everyone in this room is crazy in his own way. So if you want to use Linux to control an industrial welding laser, I have no problem with your using PREEMPT_RT."

Linus Torvalds

RT_PREEMPT from 2001 to today: Technology

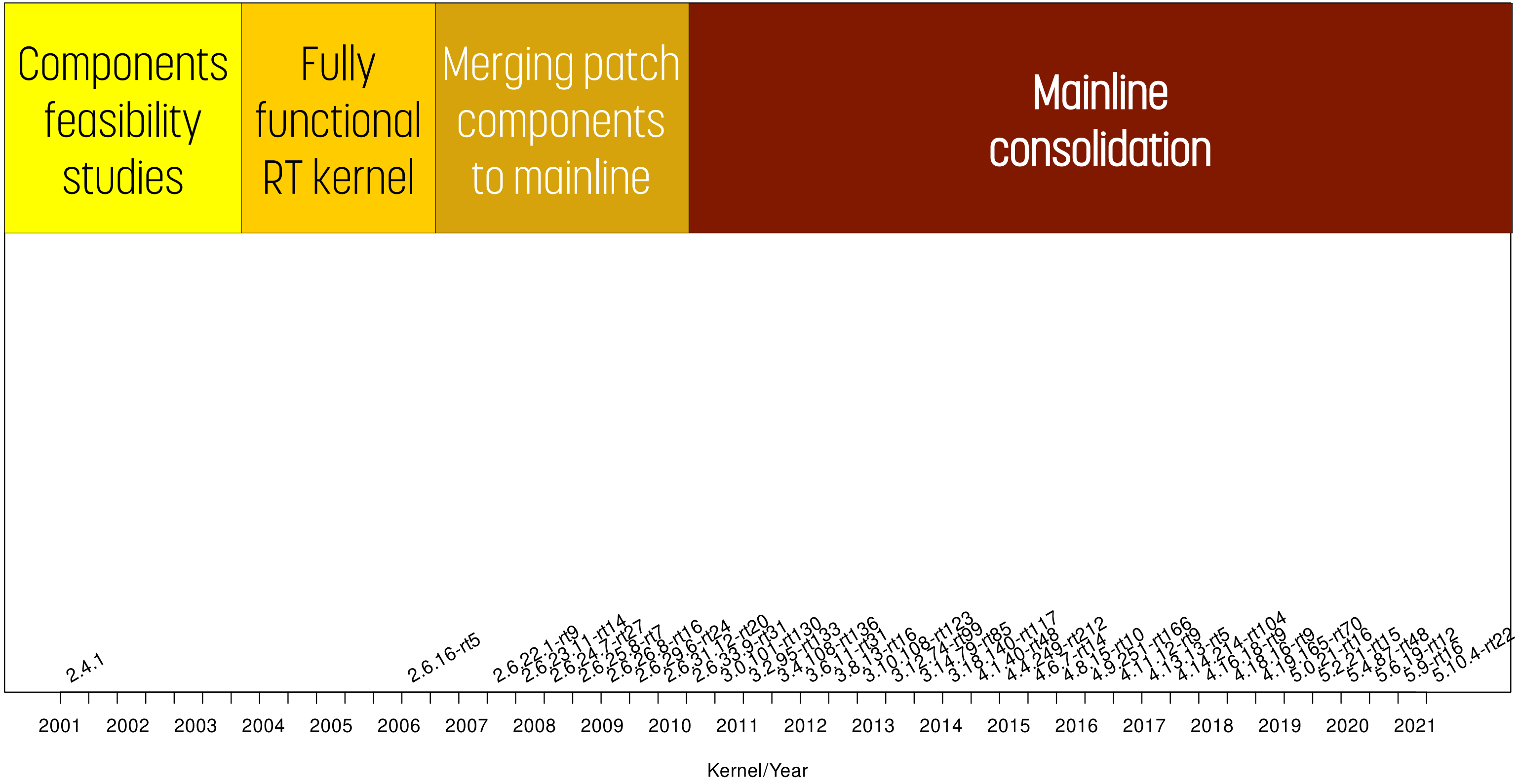


Merge patch components to mainline

Feature \ Architecture	x86	x86/64	powerpc	arm	mips	68knommu
Deterministic Scheduler	green	green	green	green	green	green
Preemption Support	green	green	green	green	green	green
PI Mutexes	green	green	green	green	green	yellow [3]
High-Resolution Timer	green	green [1]	green [1]	green [1]	green [1]	green
Preemptive Read-Copy Update	green [2]	green [2]	green [2]	green [2]	green [2]	green [2]
IRQ Threads	green [4]	green [4]	green [4]	green [4]	green [4]	yellow [3,4,5]
Raw Spinlock Annotation	green [6]	green [6]	green [6]	green [6]	green [6]	green [6]
Full Realtime Preemption Support	yellow	yellow	yellow	yellow	yellow	yellow [3]

green: Available in mainline Linux
yellow: Available when Realtime-Preempt patches applied

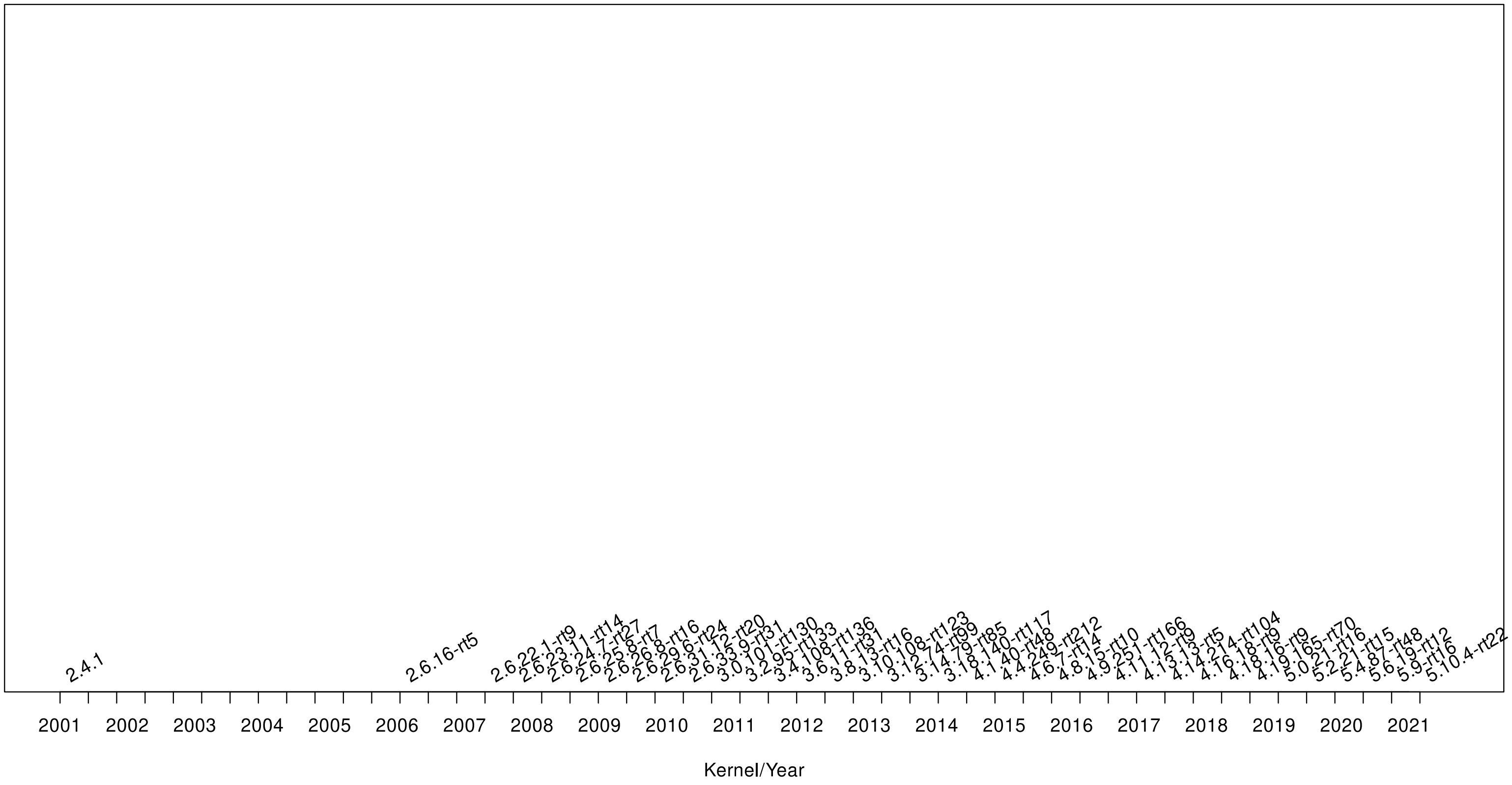
RT_PREEMPT from 2001 to today: Technology



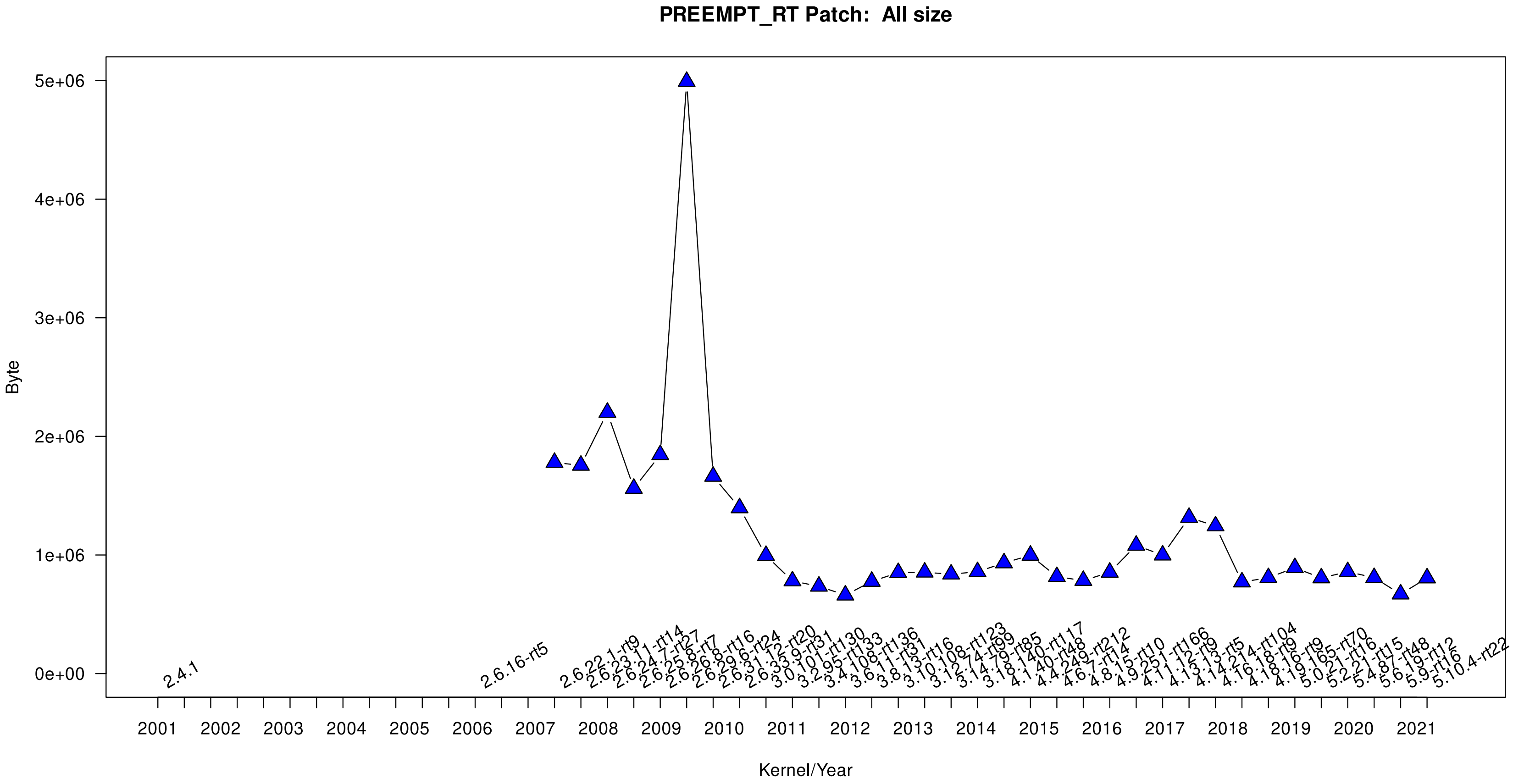
COOL COMPACT OSADL ONLINE LECTURES

OSADL Open Source Automation Development Lab eG

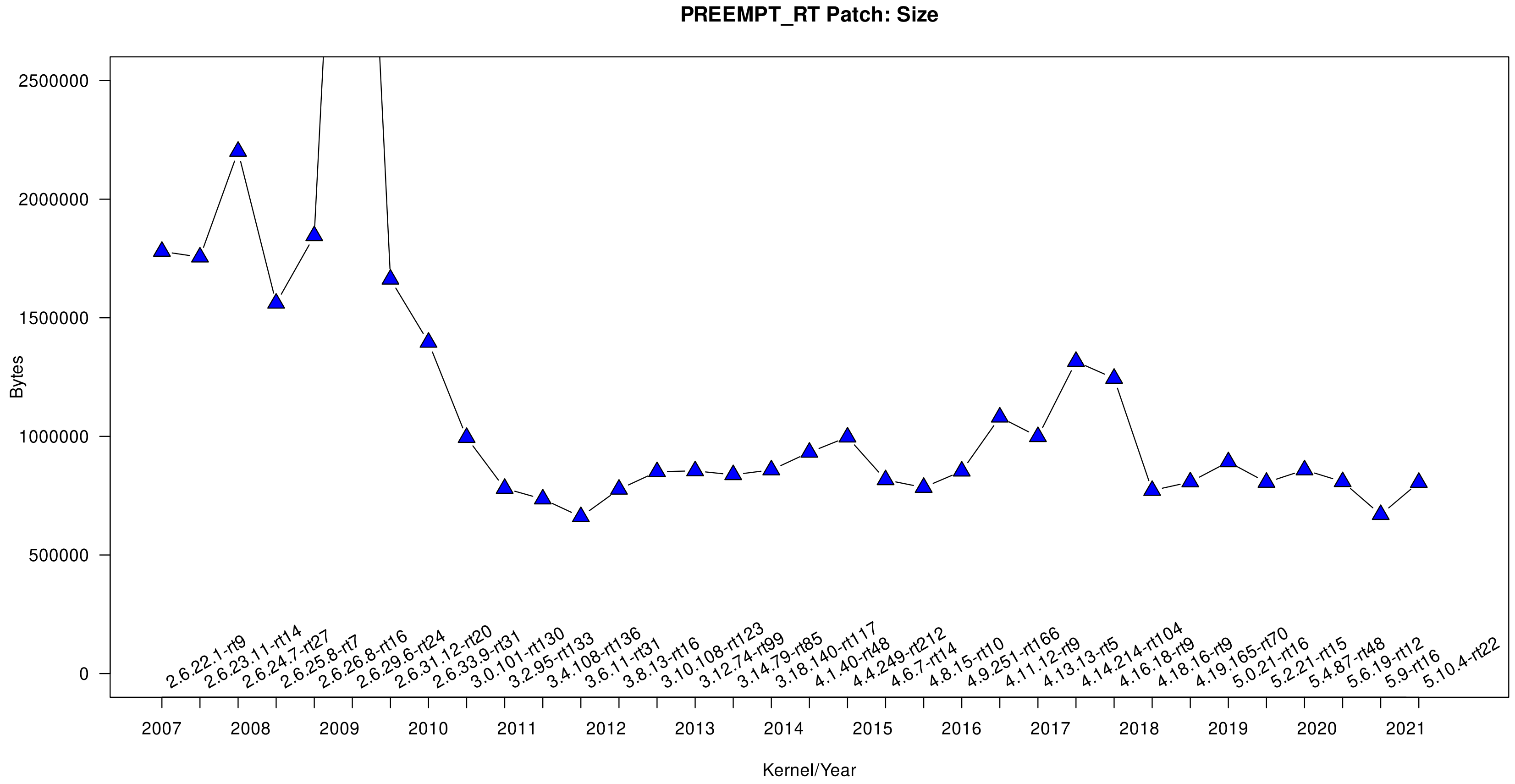
RT_PREEMPT from 2001 to today



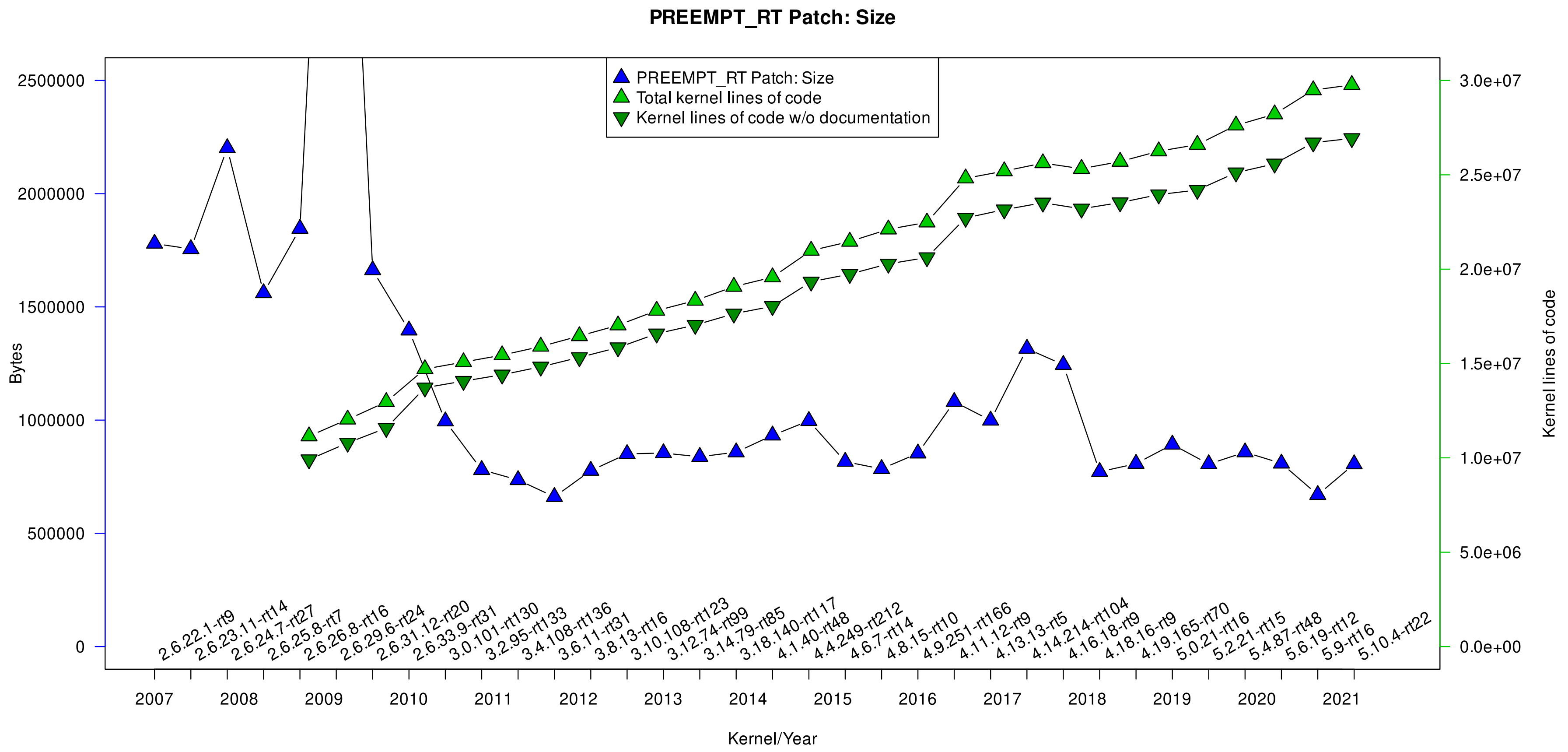
RT_PREEMPT from 2001 to today: Patch size



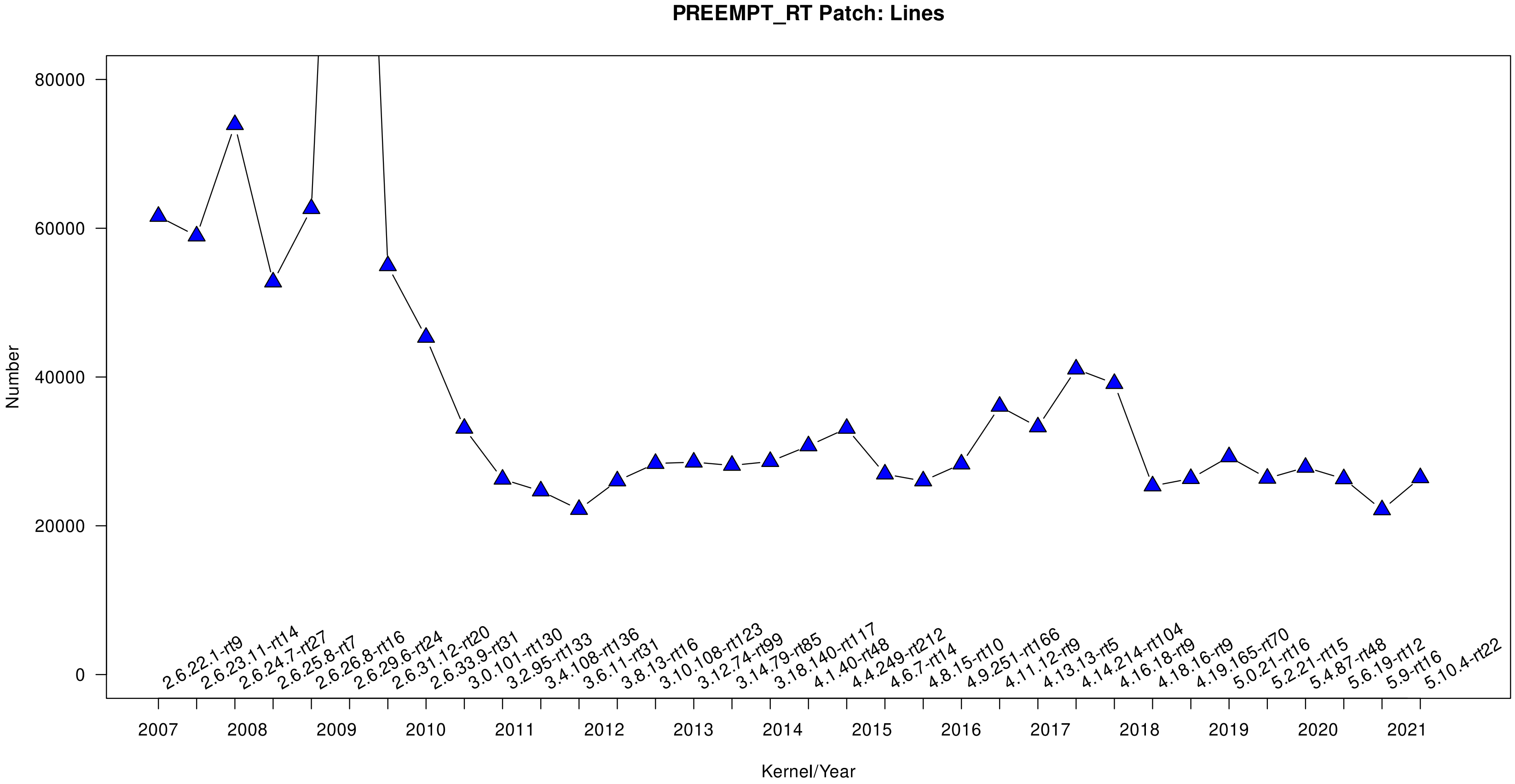
PREEMPT_RT patch size



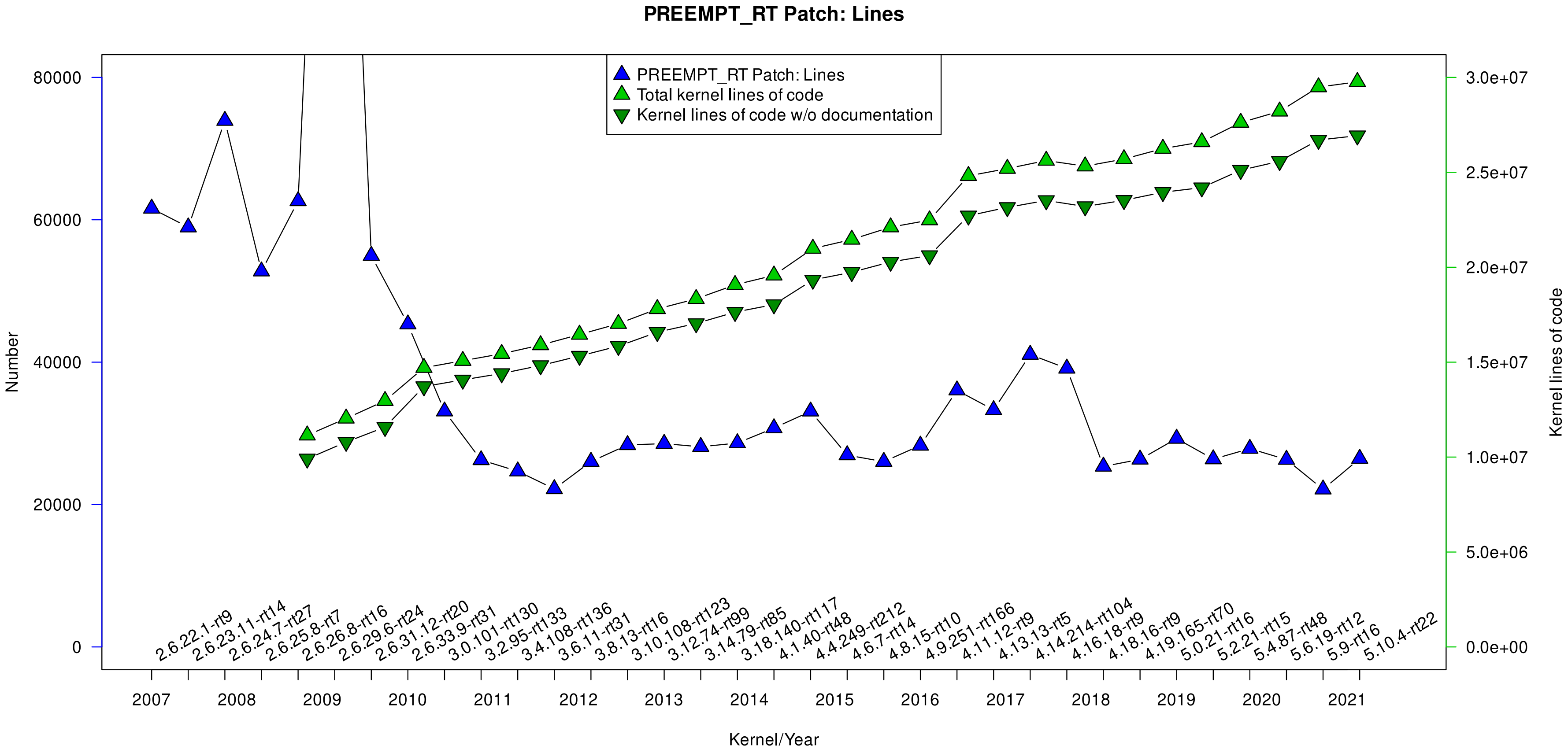
PREEMPT_RT patch size (vs. kernel size)



PREEMPT_RT lines of code

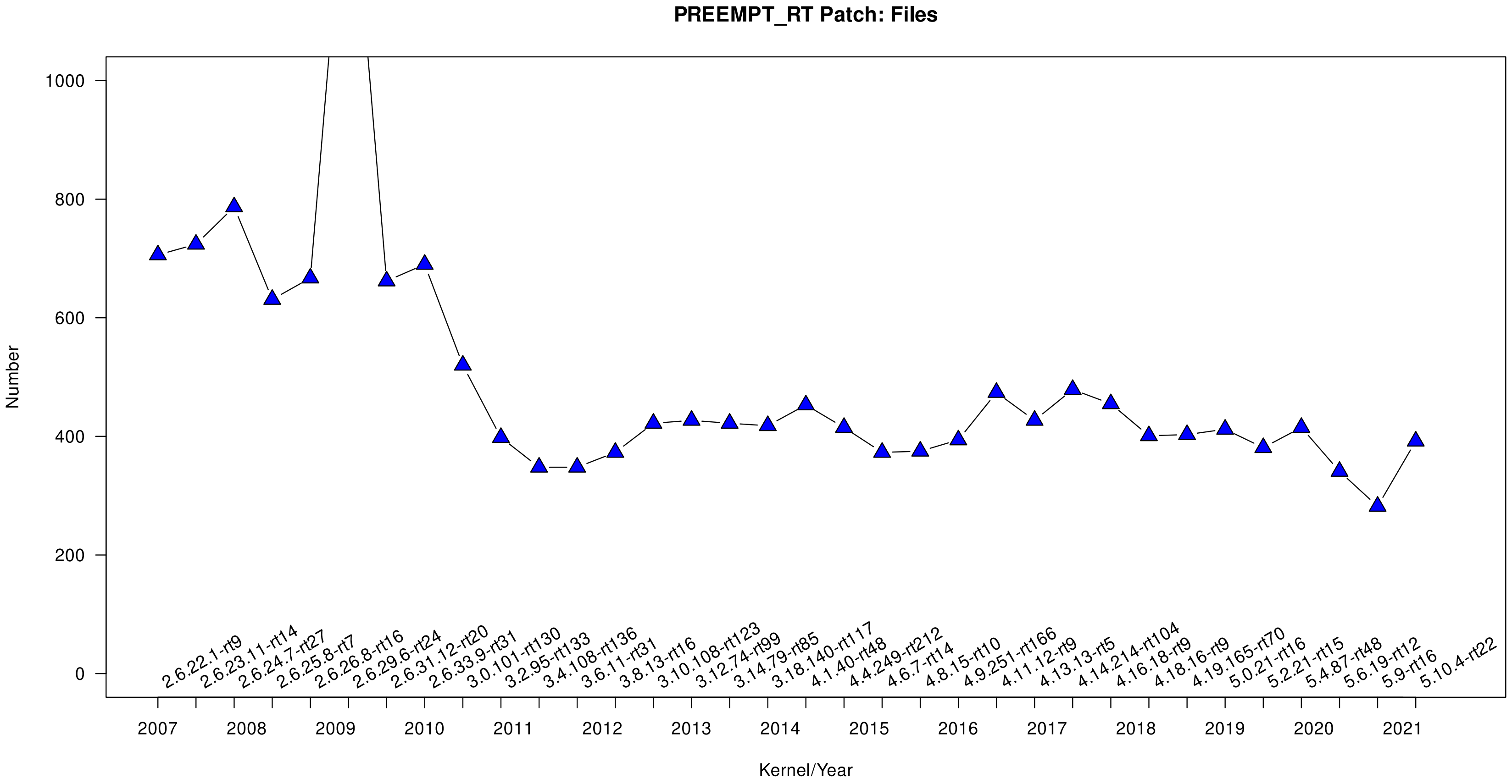


PREEMPT_RT lines of code (vs. kernel size)

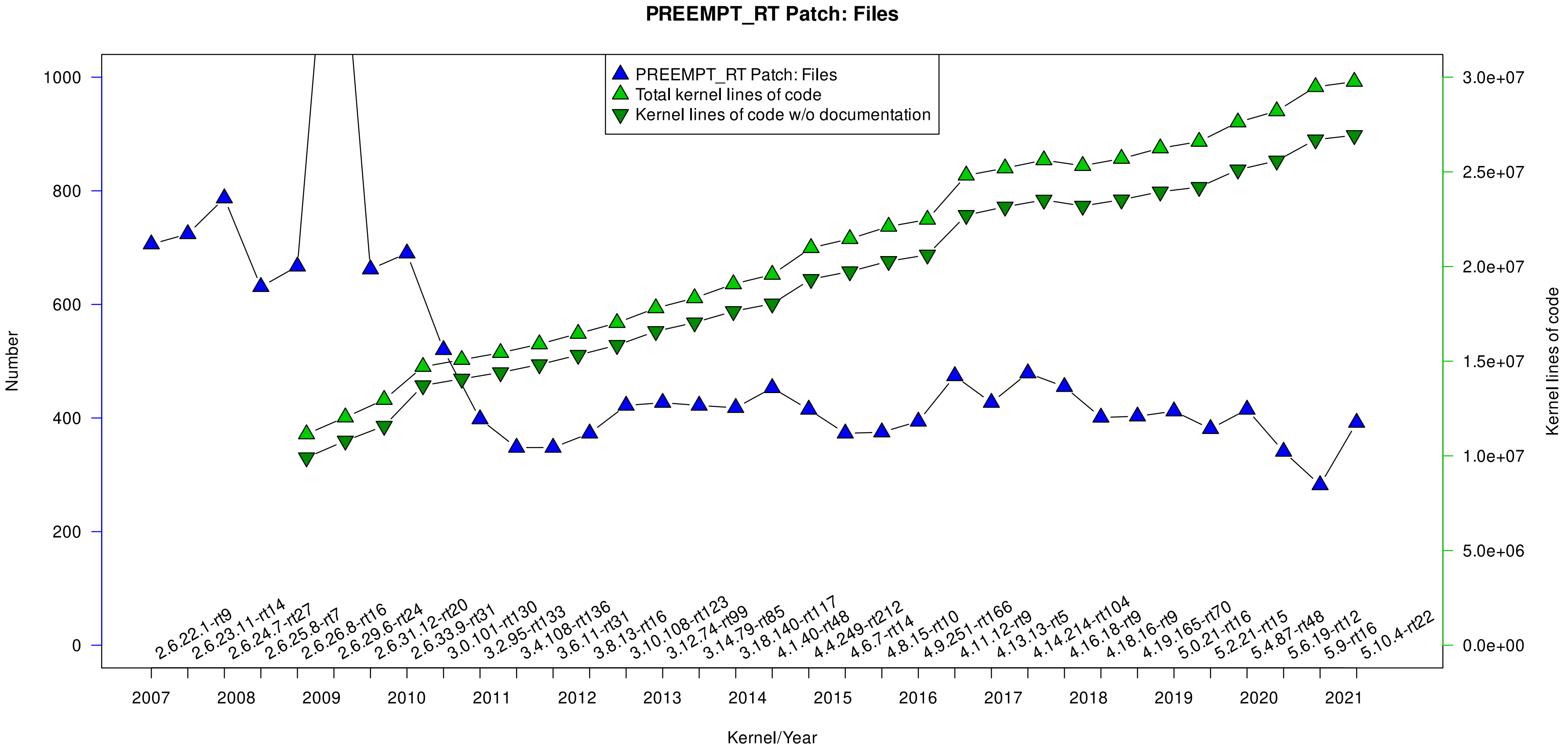


Open Source Automation Development Lab eG

PREEMPT_RT number of files



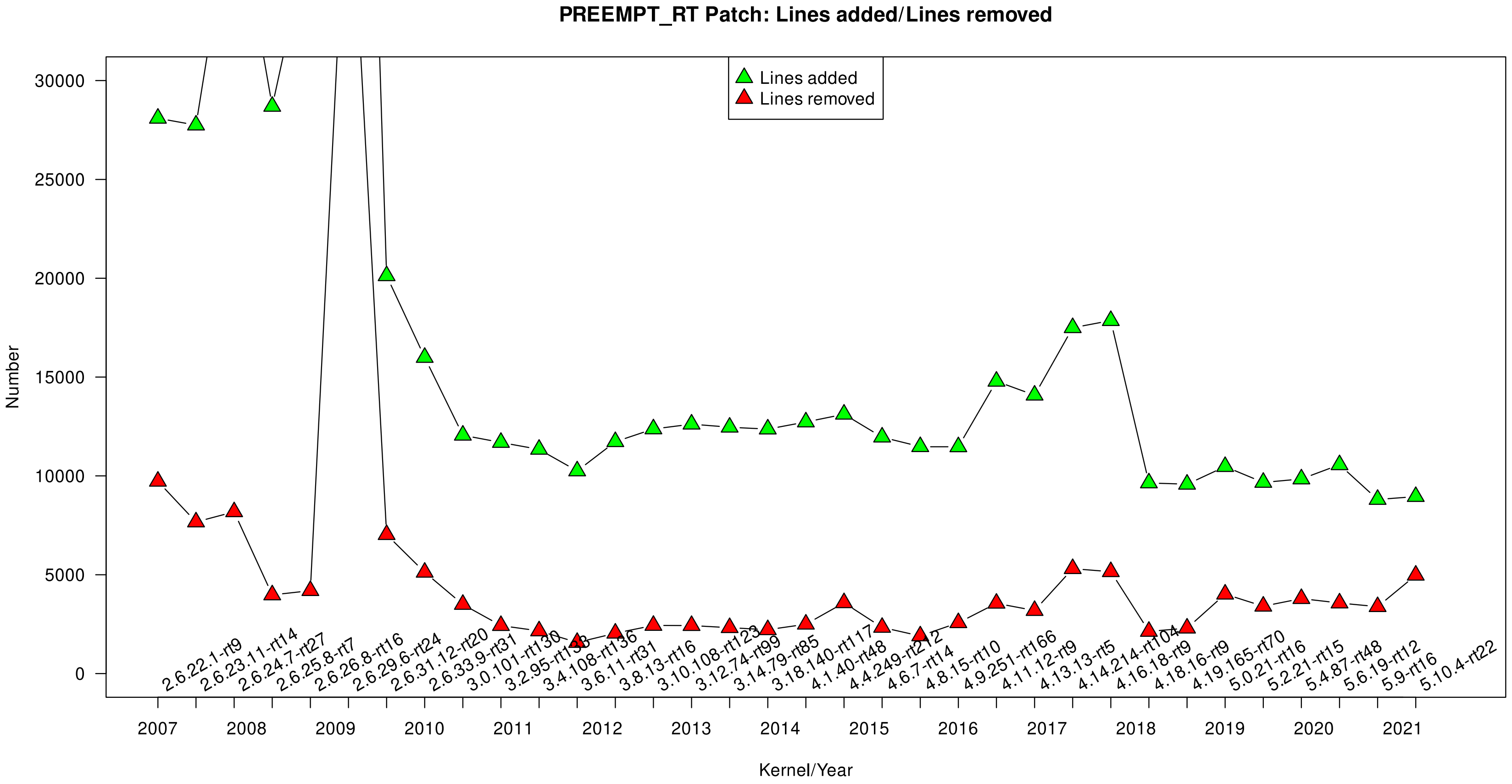
PREEMPT_RT number of files (vs. kernel size)



COMPACT OSADL ONLINE LECTURES

Open Source Automation Development Lab eG

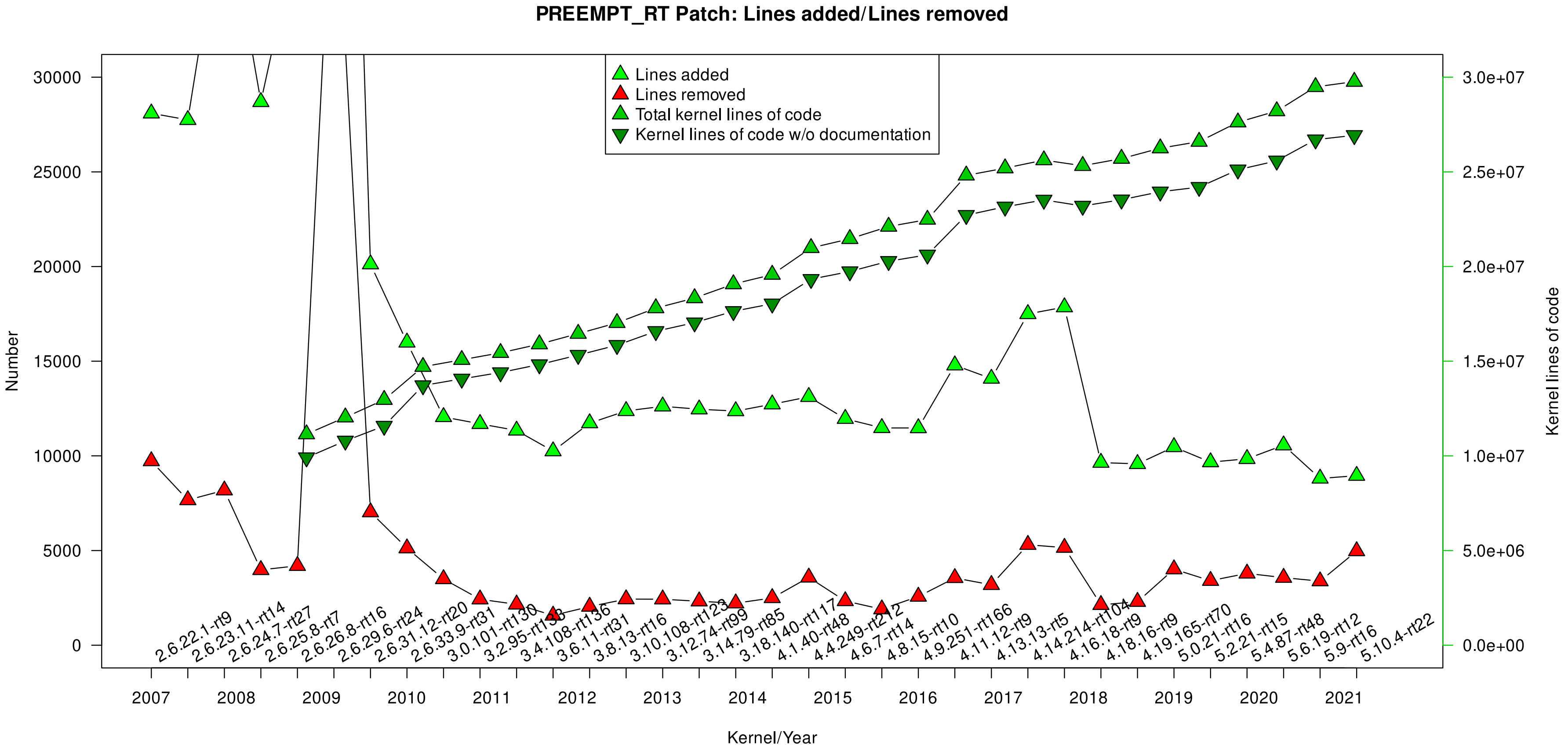
PREEMPT_RT patch activity



COMPACT OSADL ONLINE LECTURES

Open Source Automation Development Lab eG

PREEMPT_RT patch activity (vs. kernel size)



Mainline consolidation

- Many mainline components could not simply be made "real-time aware", but needed to be redesigned.
- Example: CPU Hotplug

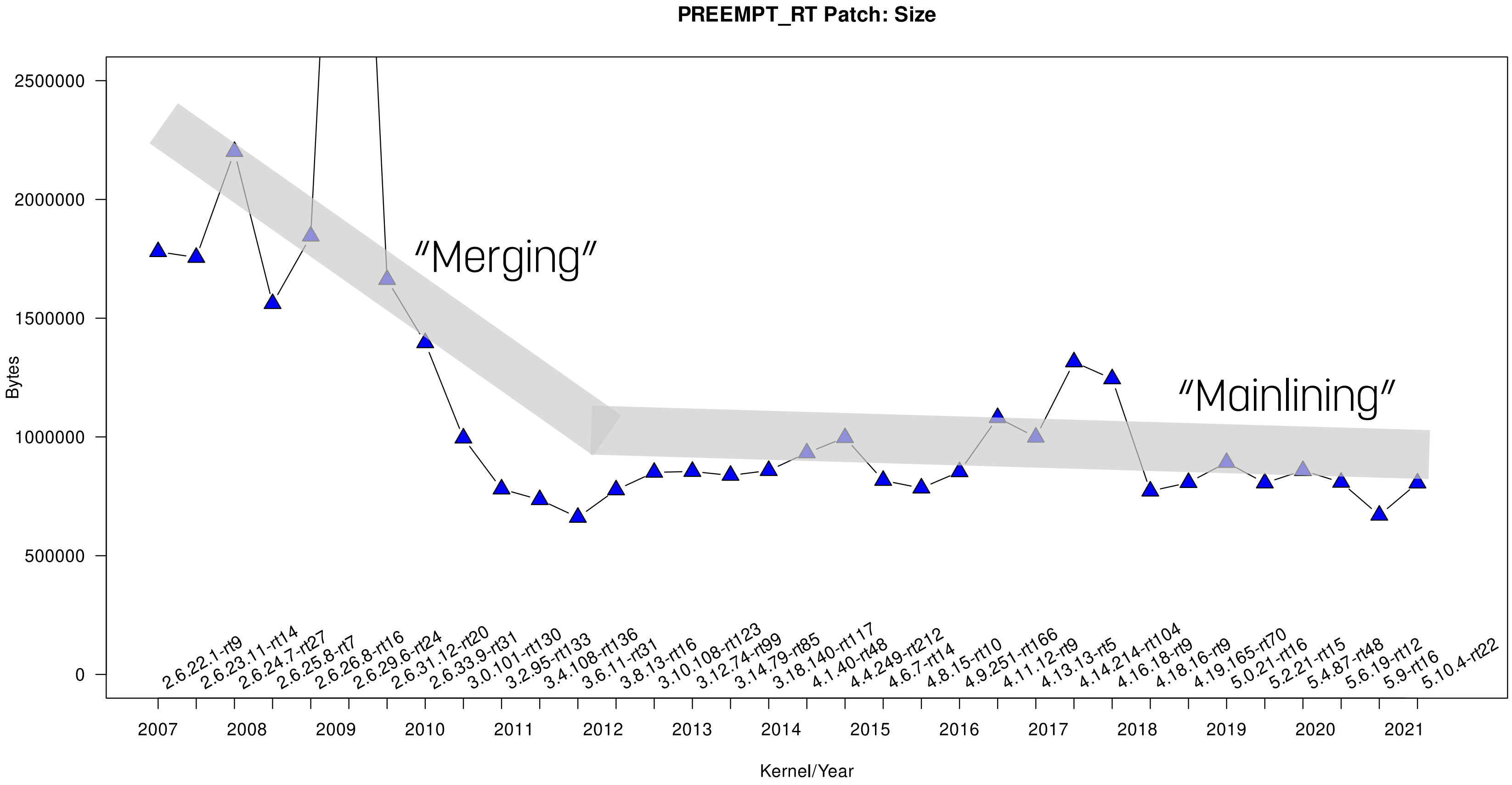
CPU Hotplug (1)

- CPU Hotplug consists of a number of steps to bring a core up or down that must be executed in a defined order.
- CPU Hotplug is not "nice to have", but always used at boot and at shutdown time.
- CPU Hotplug did not work well with PREEMPT_RT, since the defined execution order of the hotplug steps was no longer ensured. Therefore, PREEMPT_RT patched systems often failed to reboot.

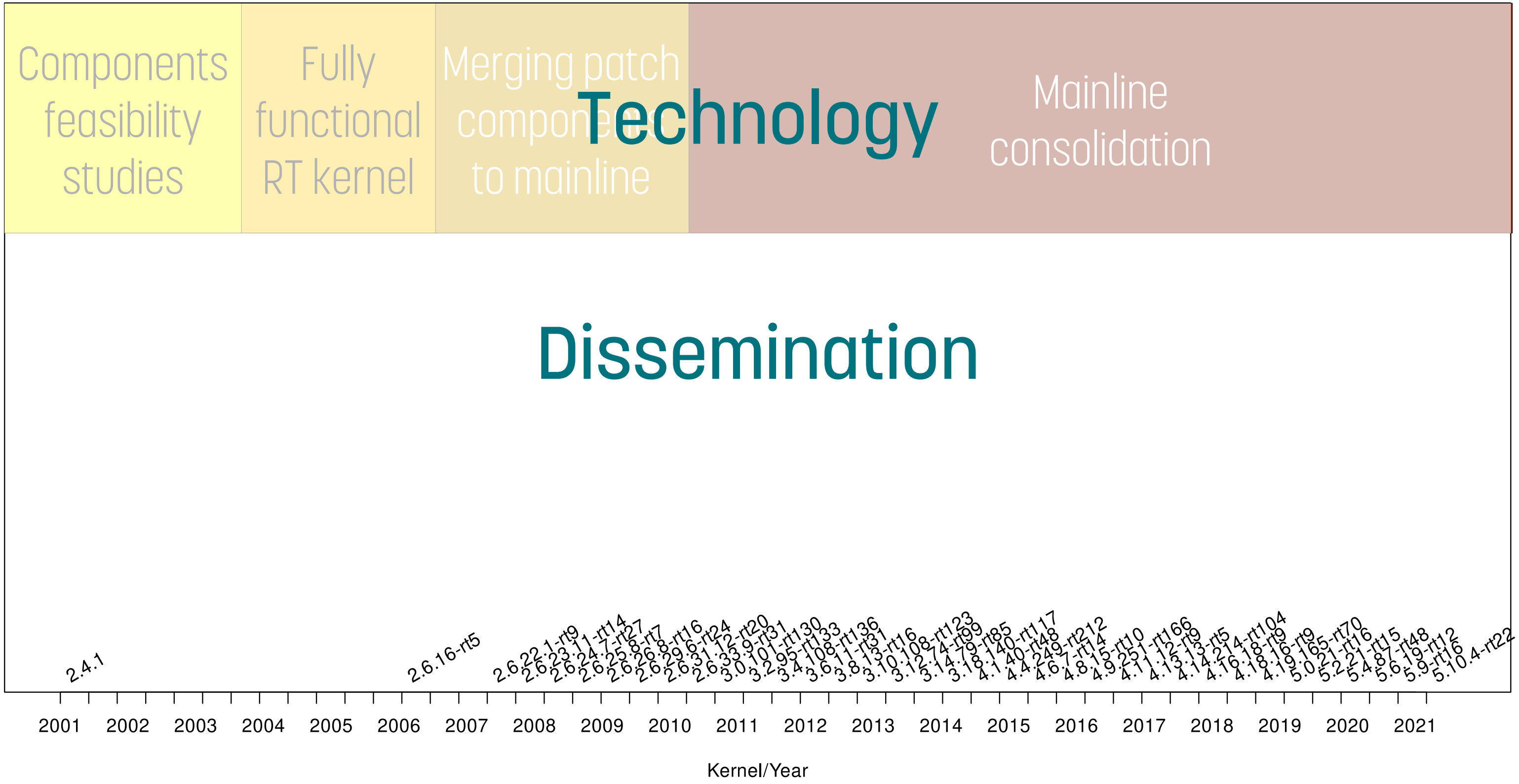
CPU Hotplug (2)

- The CPU Hotplug code is in mainline, not in the code of the PREEMPT_RT patch.
- The complete overhaul of the Linux CPU Hotplug subsystem was tedious and time consuming and done under the heading "RT_PREEMPT" mainlining.
- When the work was finished, both mainline kernel and PREEMPT_RT improved, but the patch size was not affected.

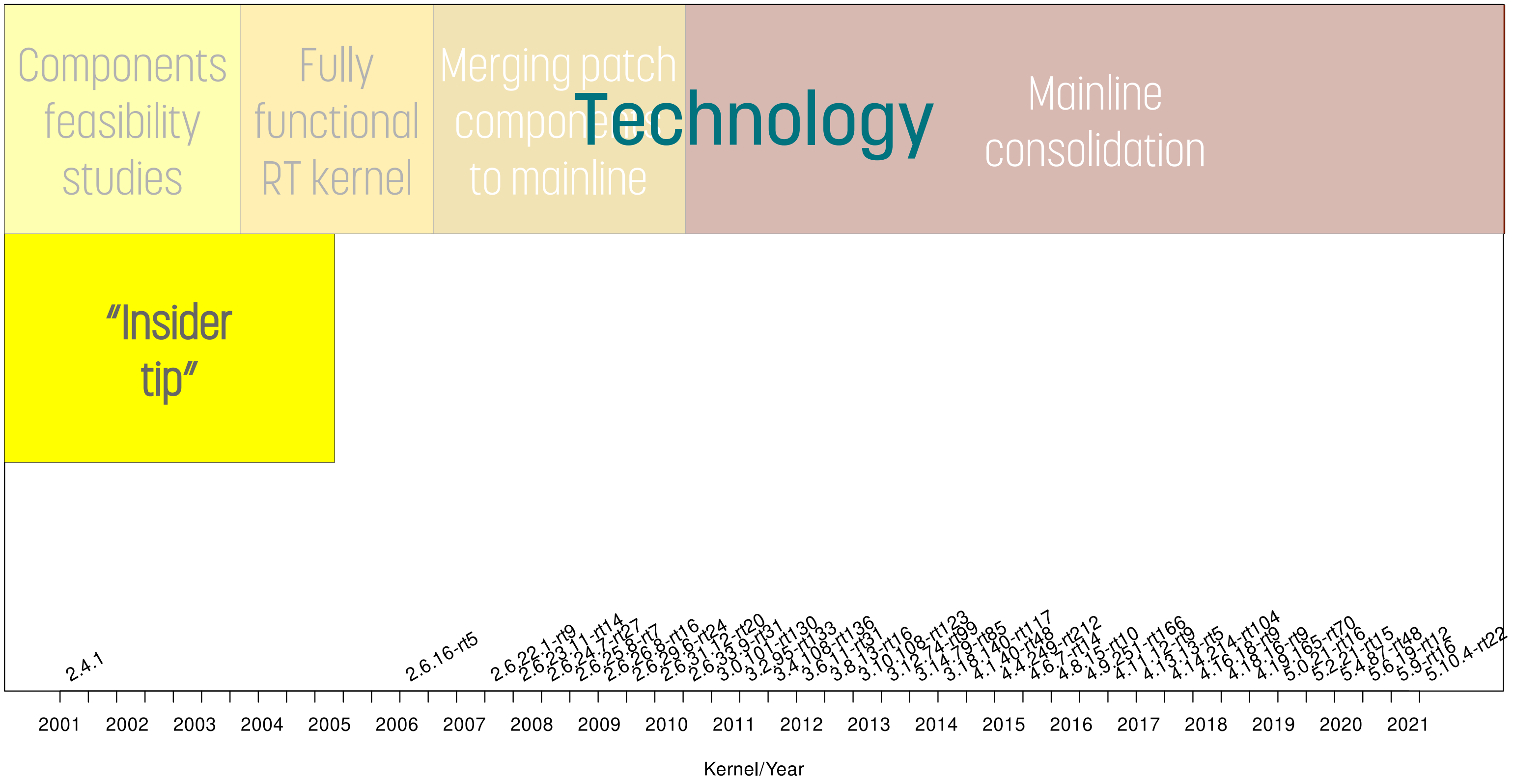
PREEMPT_RT patch size



RT_PREEMPT from 2001 to today: Dissemination



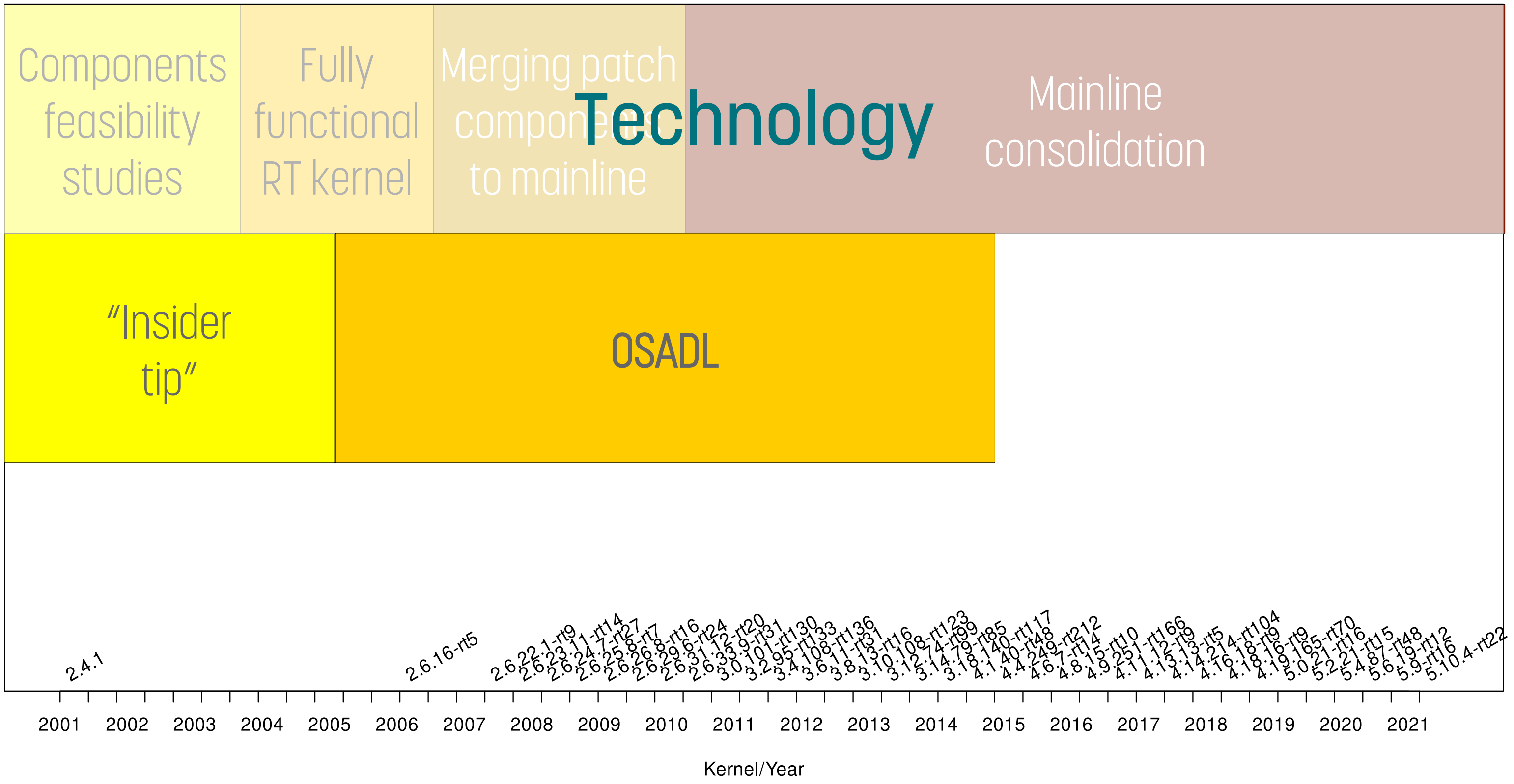
RT_PREEMPT from 2001 to today: Dissemination



Why “Insider tip”?

- Real-time systems were initially required, among other, by automation industry.
- Open Source software such as Linux did not have a particularly good reputation in automation industry.
- Community development strategies such as early and frequent releases and software patches were perceived as strange by automation industry.

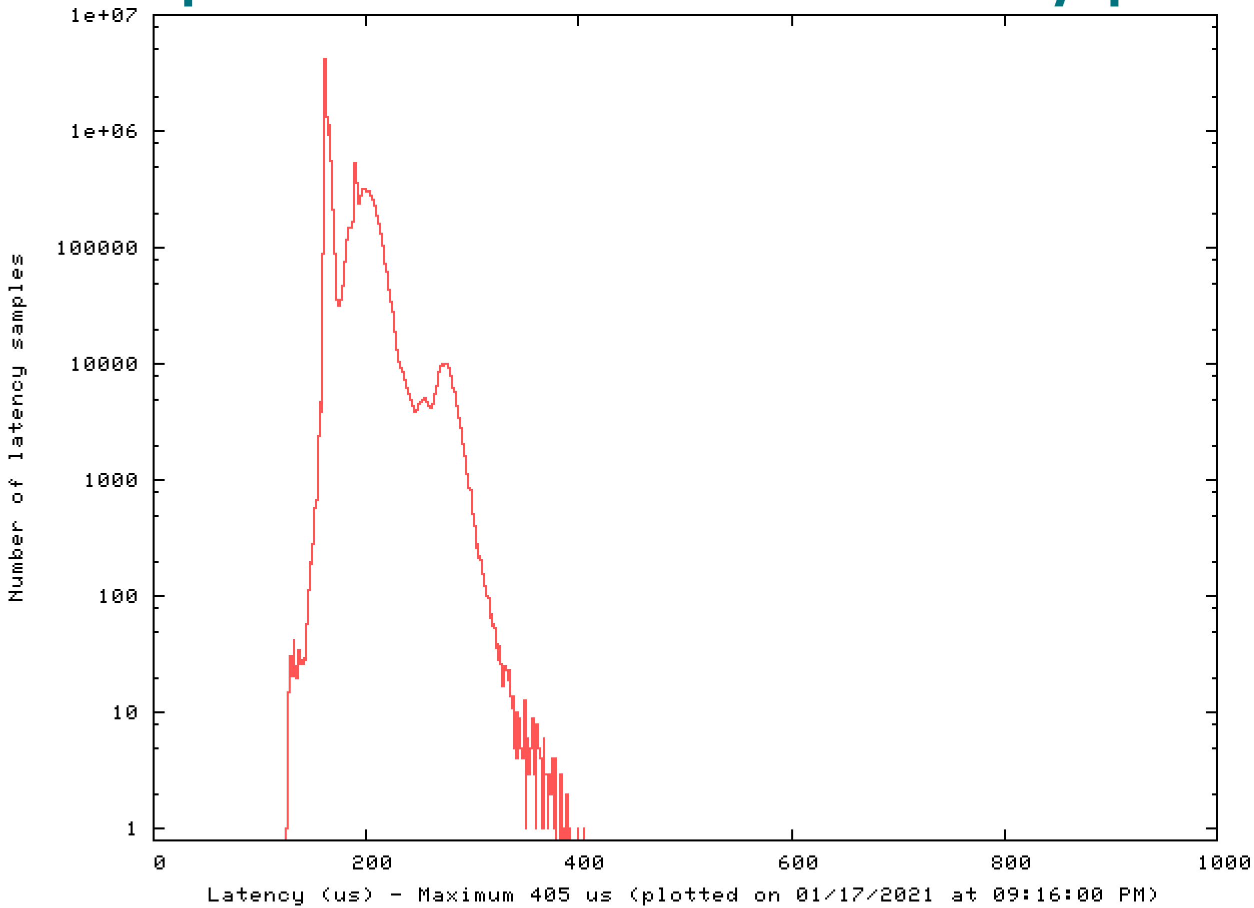
RT_PREEMPT from 2001 to today: Dissemination



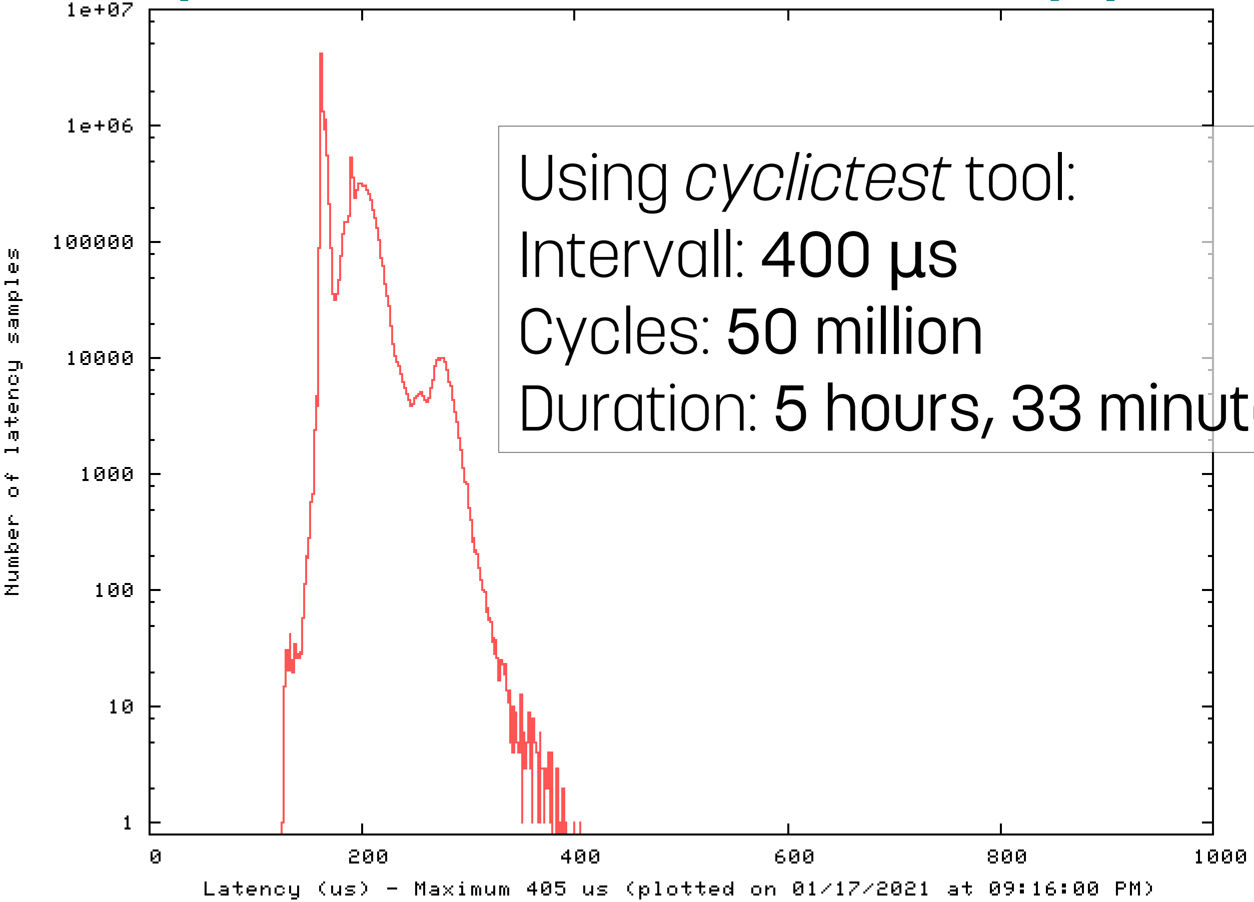
OSADL started with latency tests

- It was clear that latency tests cannot replace path analysis.
- However, although real-time compliance cannot be experimentally established, it can be proven that a system is not real-time compliant.
- So, in a first step latency measurements were performed to detect sources of unexpected latencies and remove them.

Example of a standard latency plot

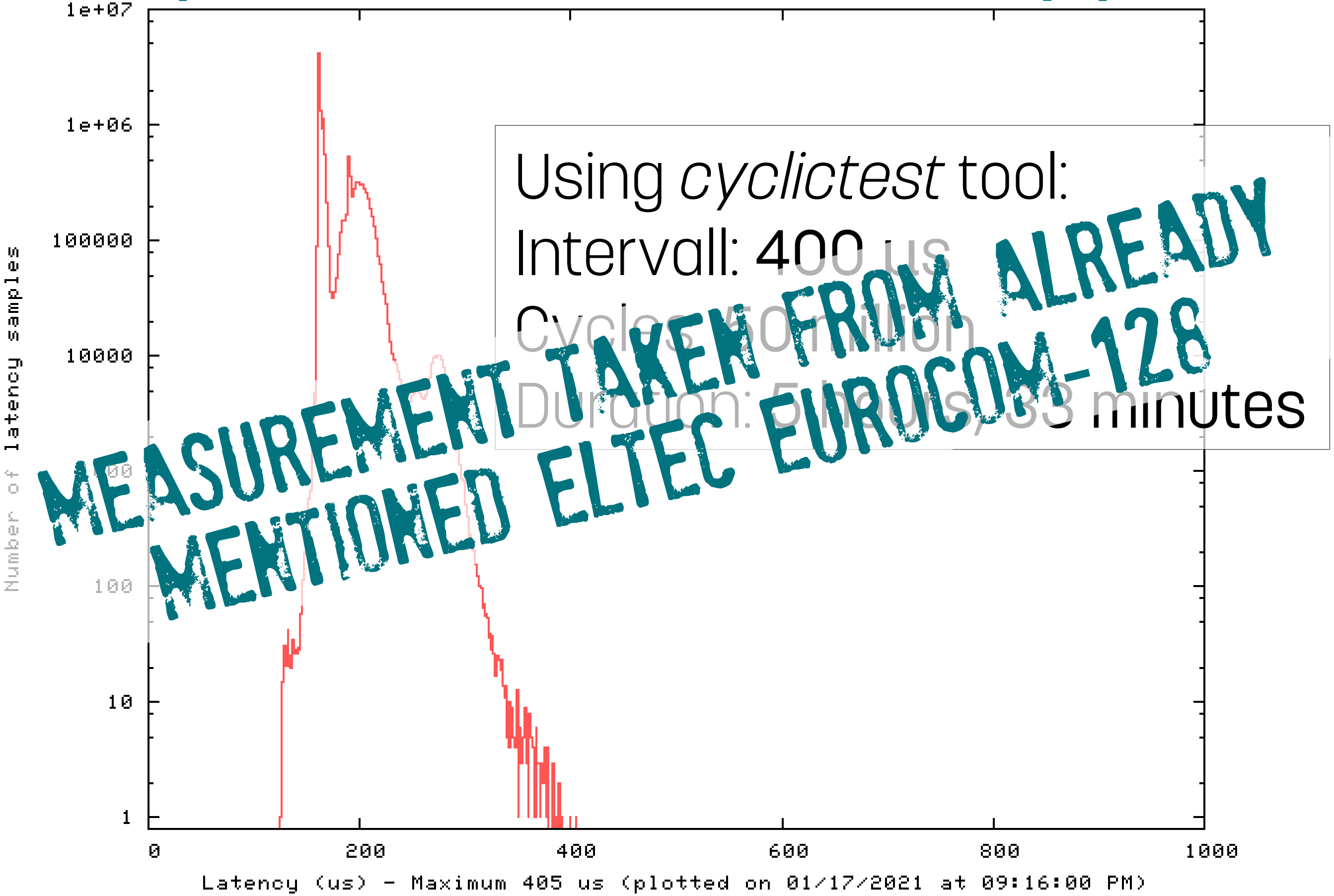


Example of a standard latency plot

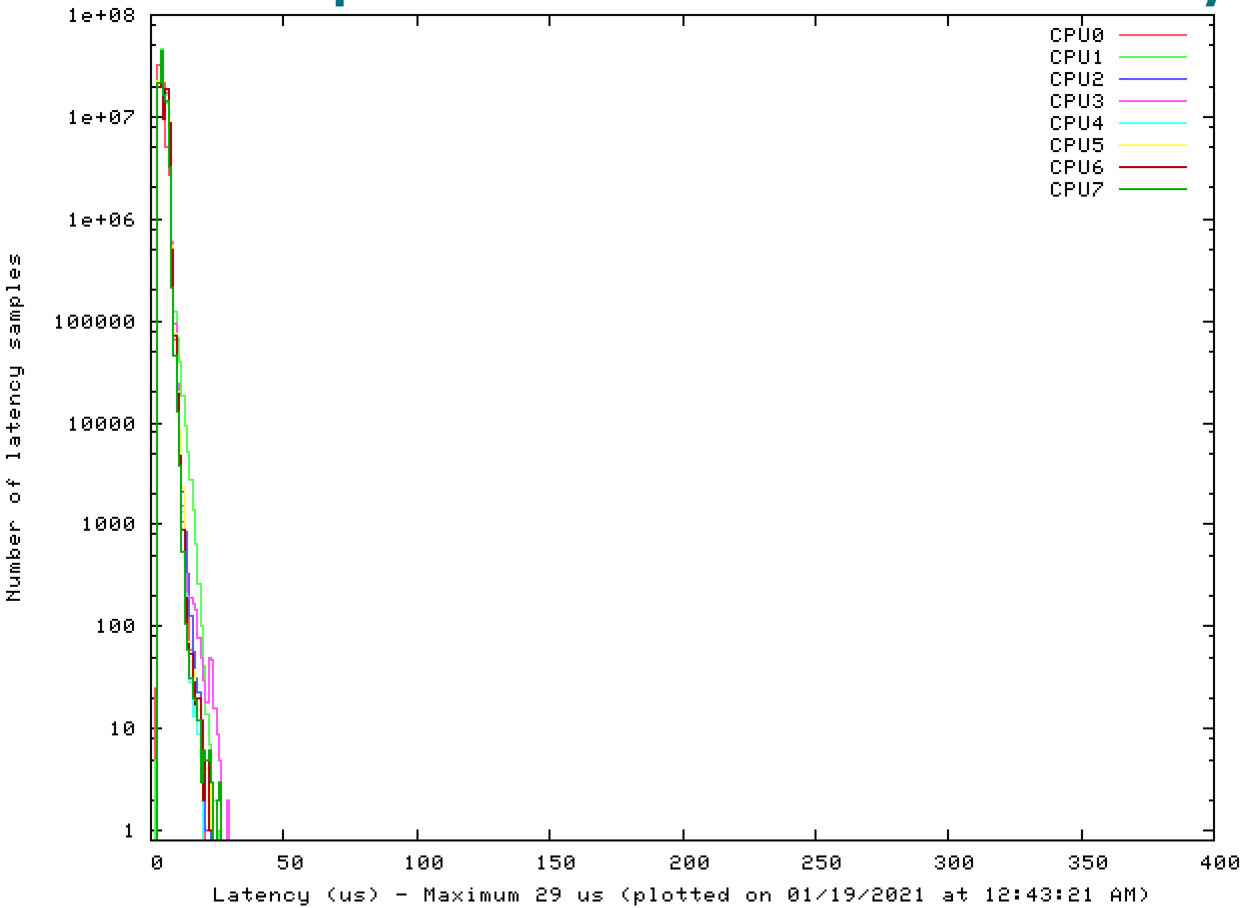


Using *cyclictest* tool:
Intervall: **400 µs**
Cycles: **50 million**
Duration: **5 hours, 33 minutes**

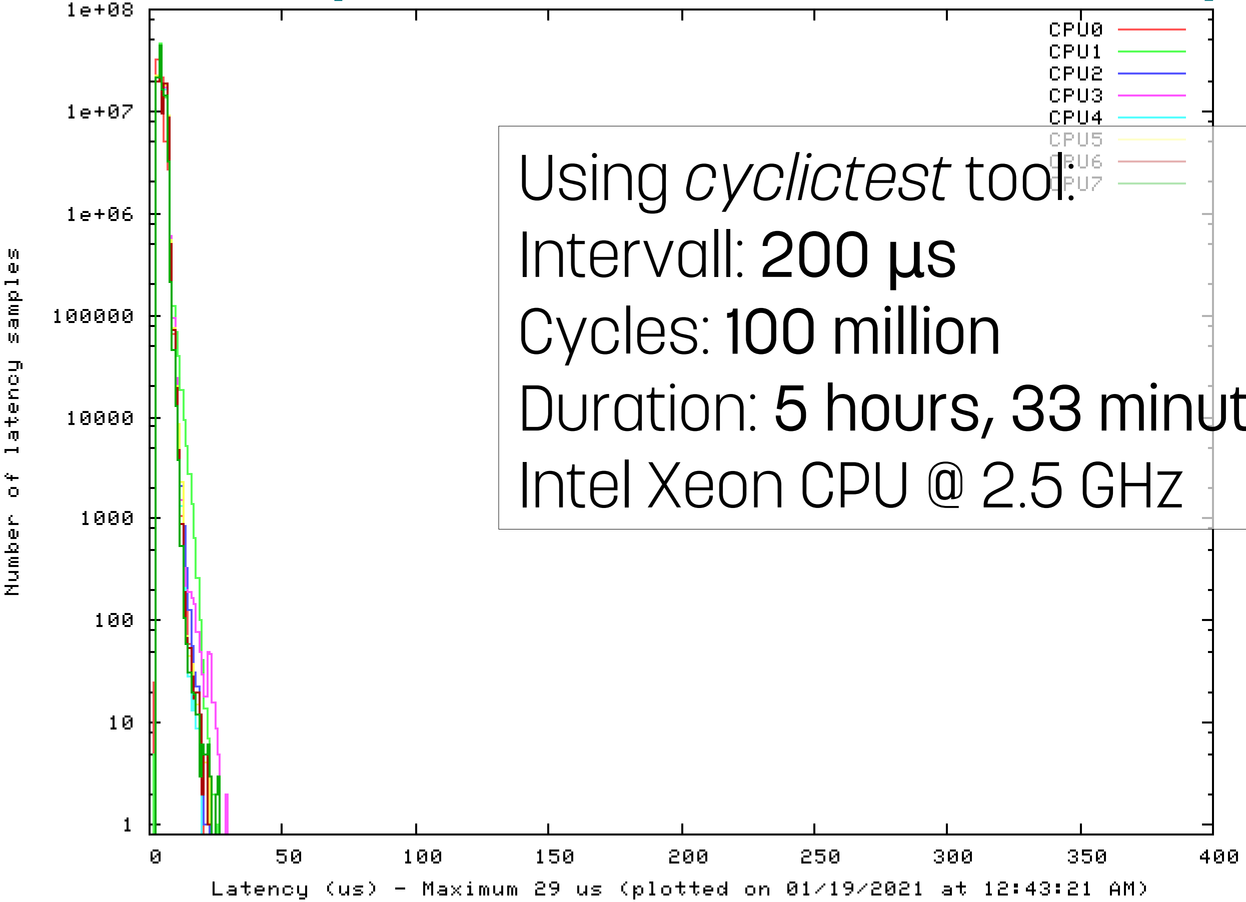
Example of a standard latency plot



Another example of a standard latency plot

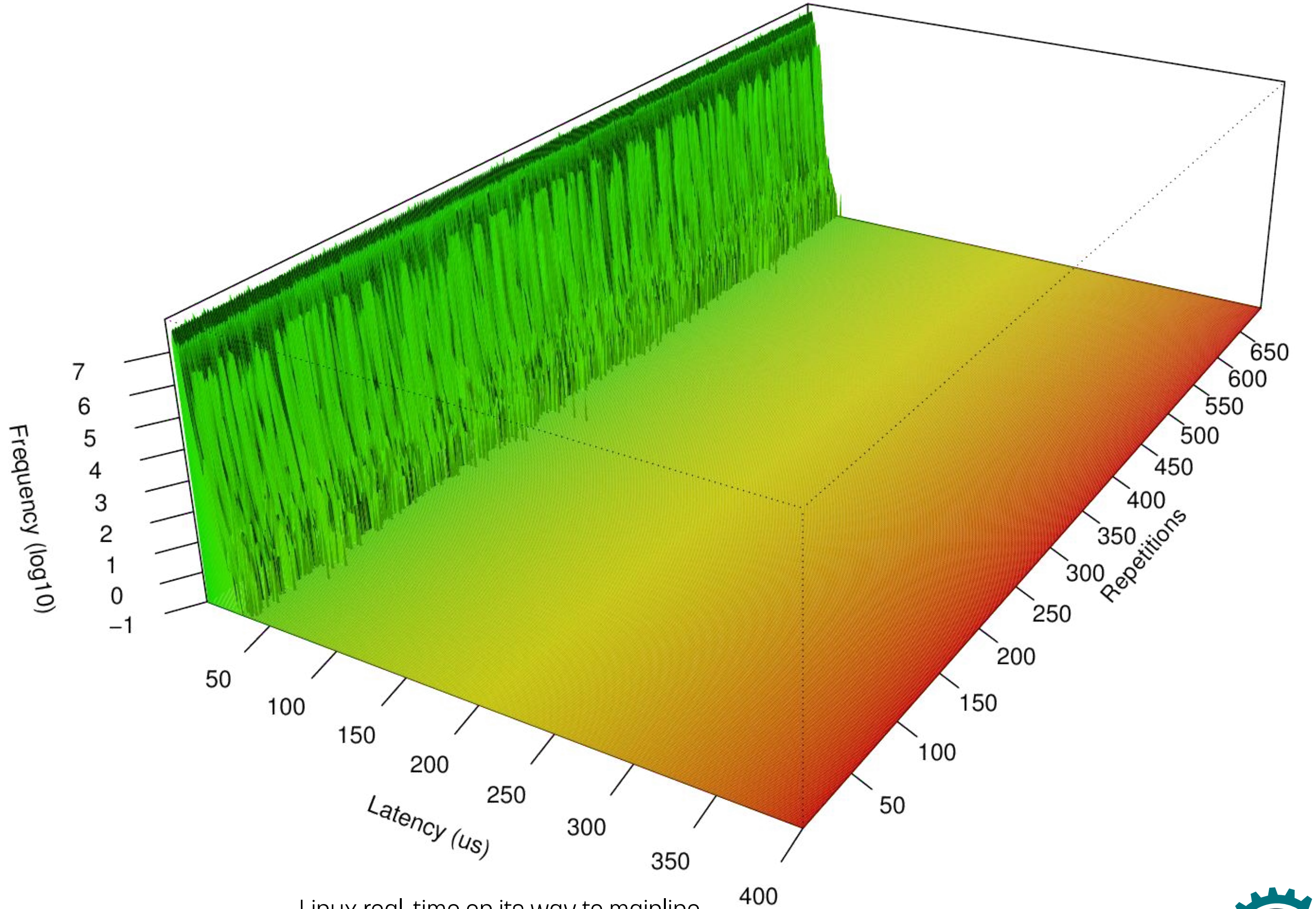


Another example of a standard latency plot



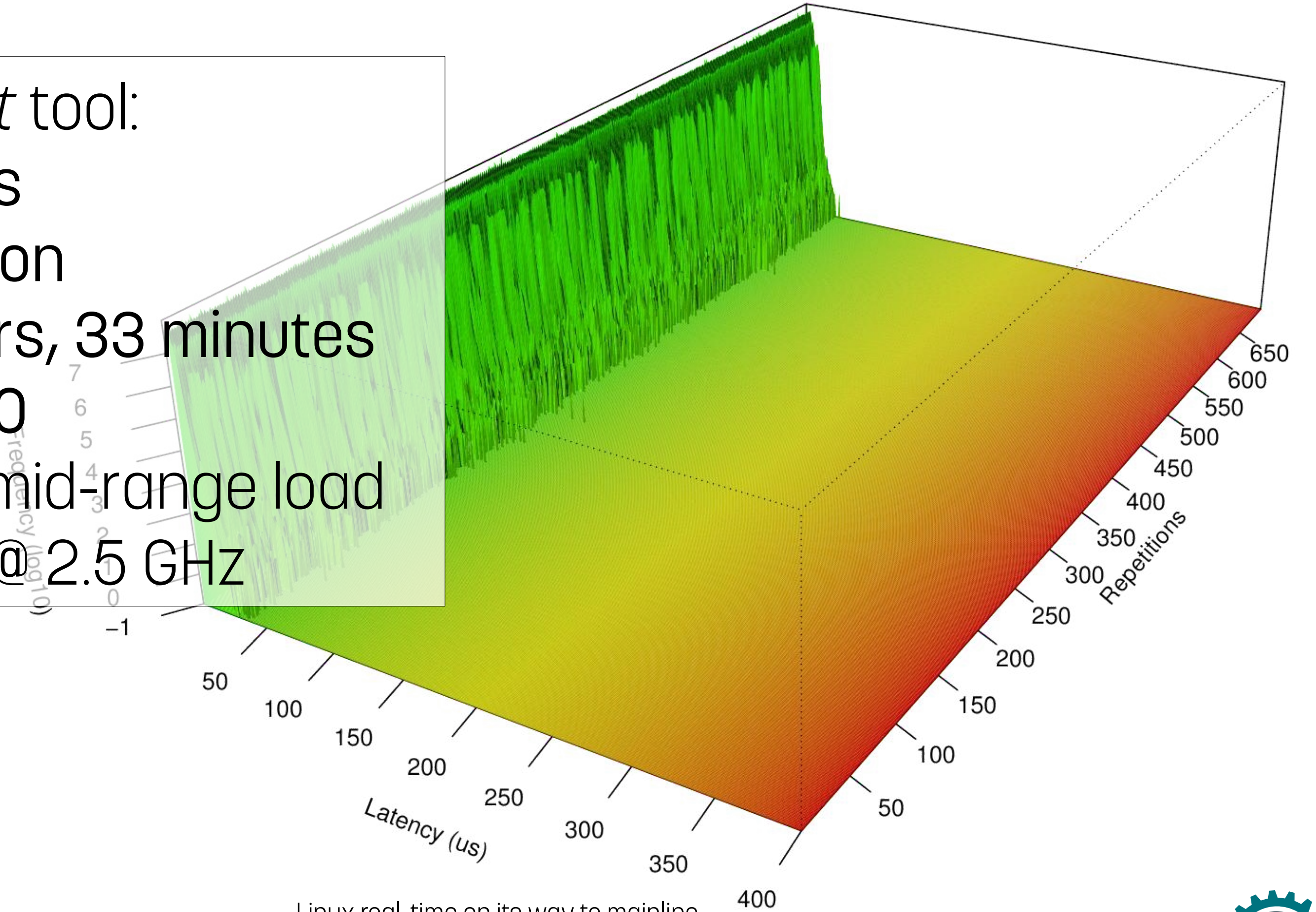
Using *cyclictest* tool:
Intervall: **200 µs**
Cycles: **100 million**
Duration: **5 hours, 33 minutes**
Intel Xeon CPU @ 2.5 GHz

Example of repeated measurements

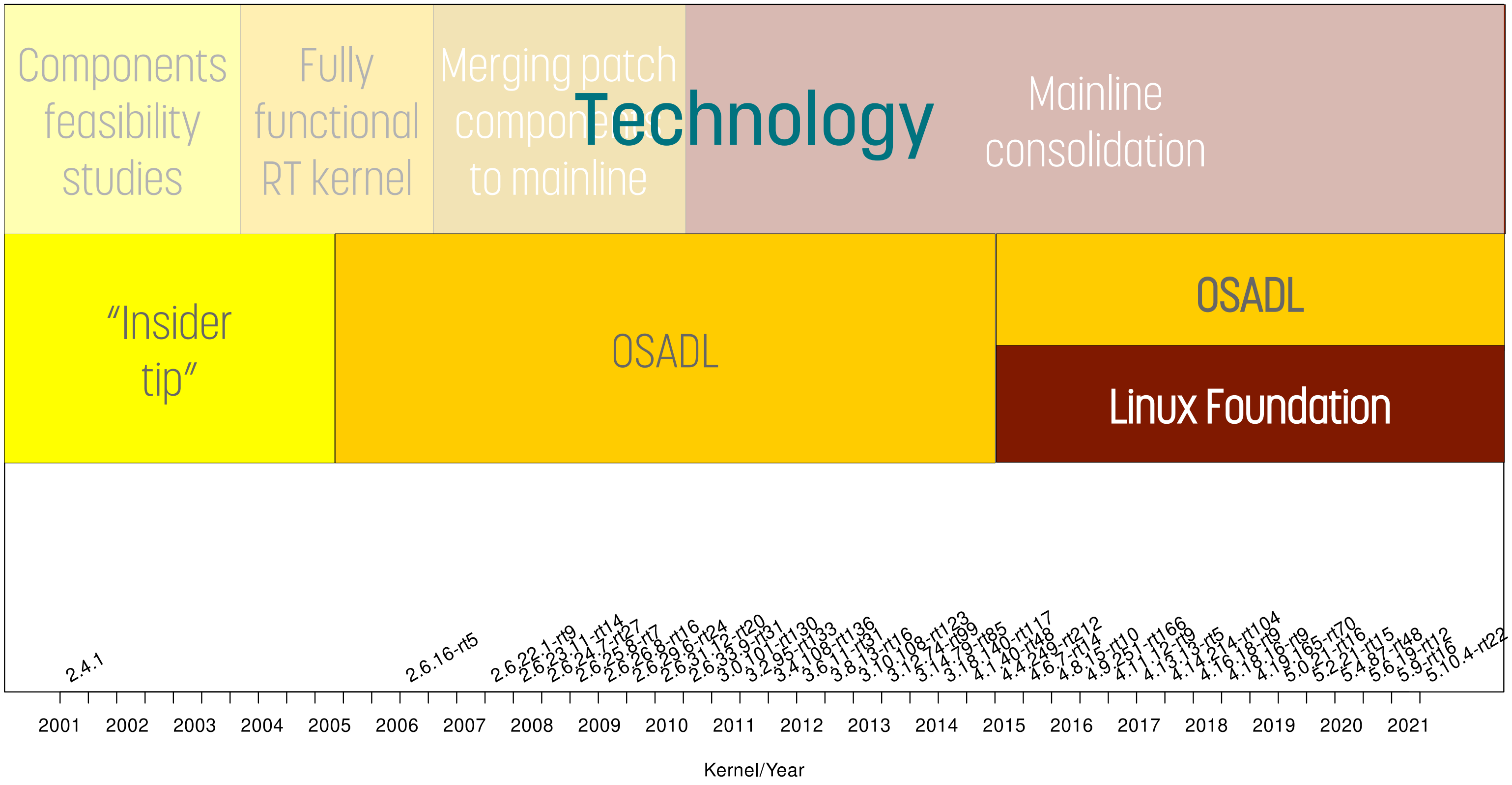


Example of repeated measurements

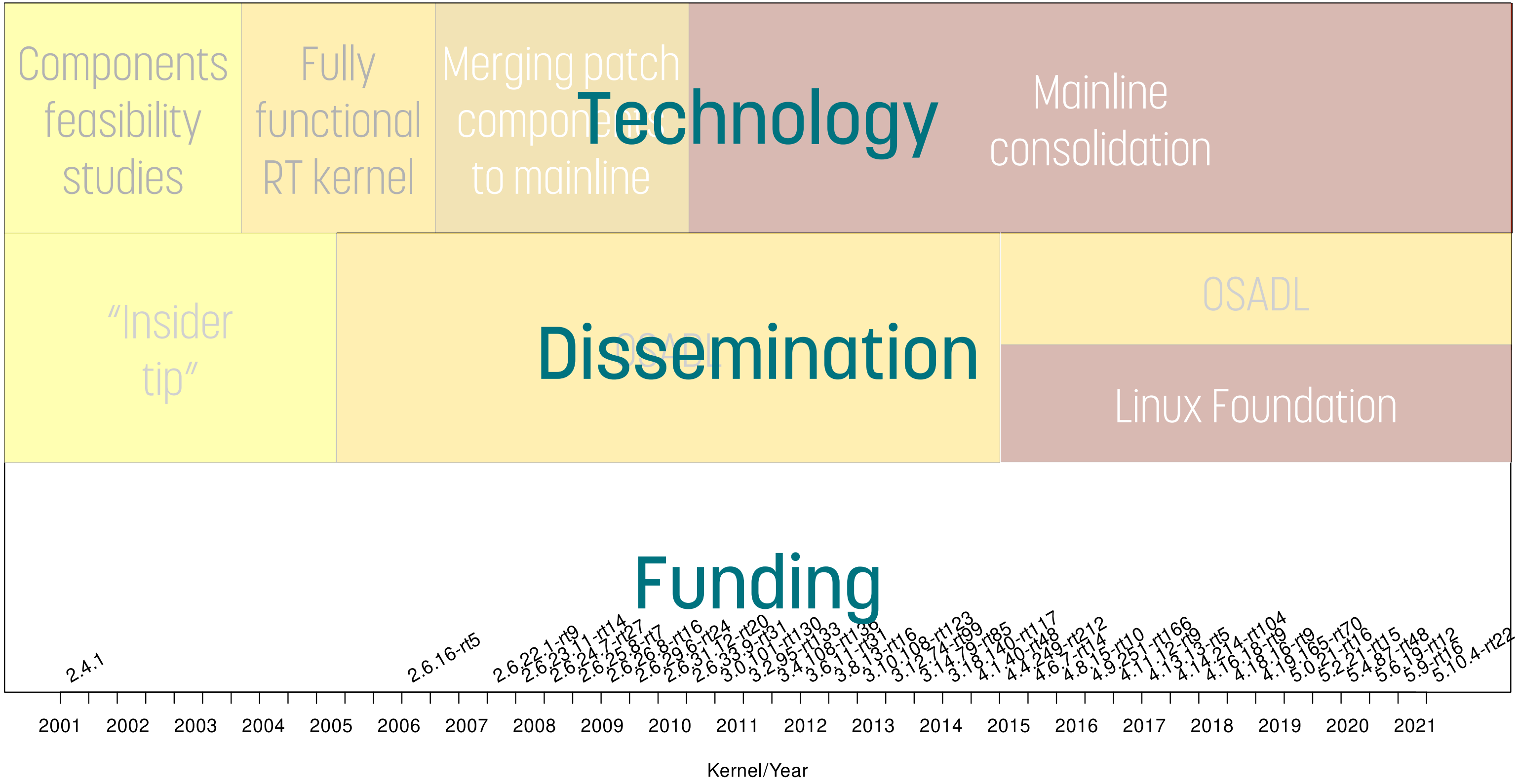
Using *cyclictest* tool:
Intervall: **200 µs**
Cycles: **100 million**
Duration: **5 hours, 33 minutes**
Runs: About **700**
Mixed idle and mid-range load
Intel Xeon CPU @ 2.5 GHz



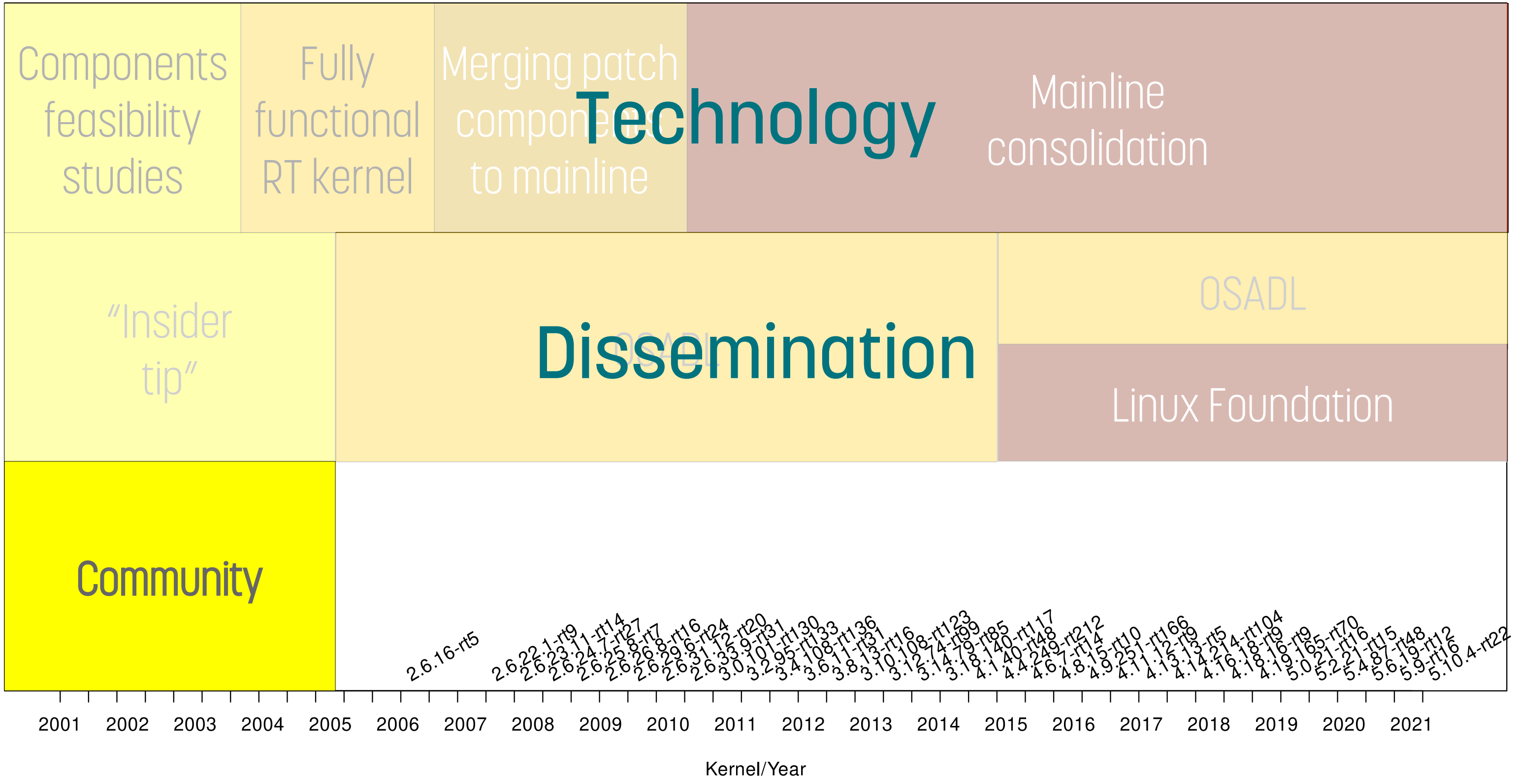
RT_PREEMPT from 2001 to today: Dissemination



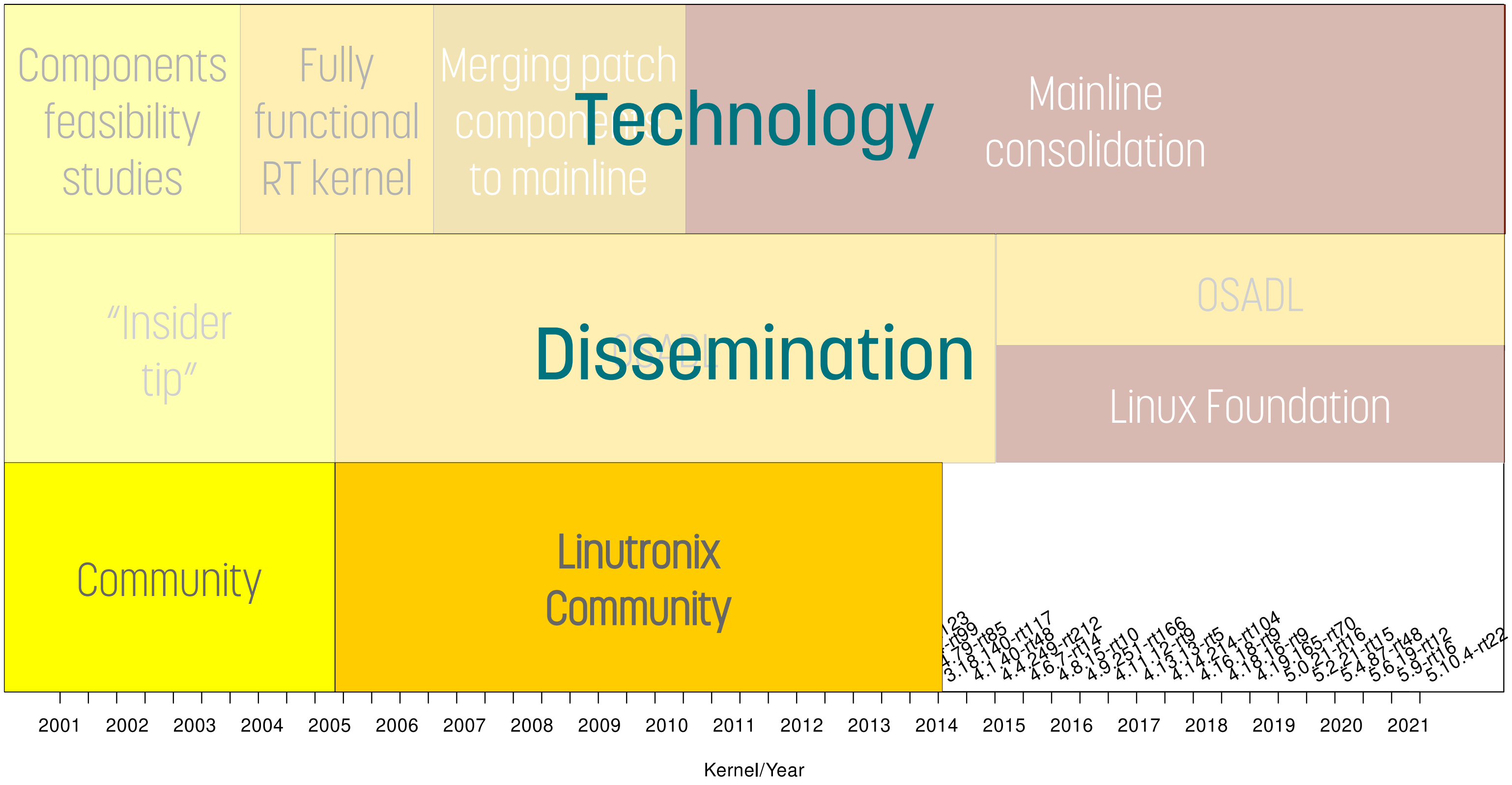
RT_PREEMPT from 2001 to today: Funding



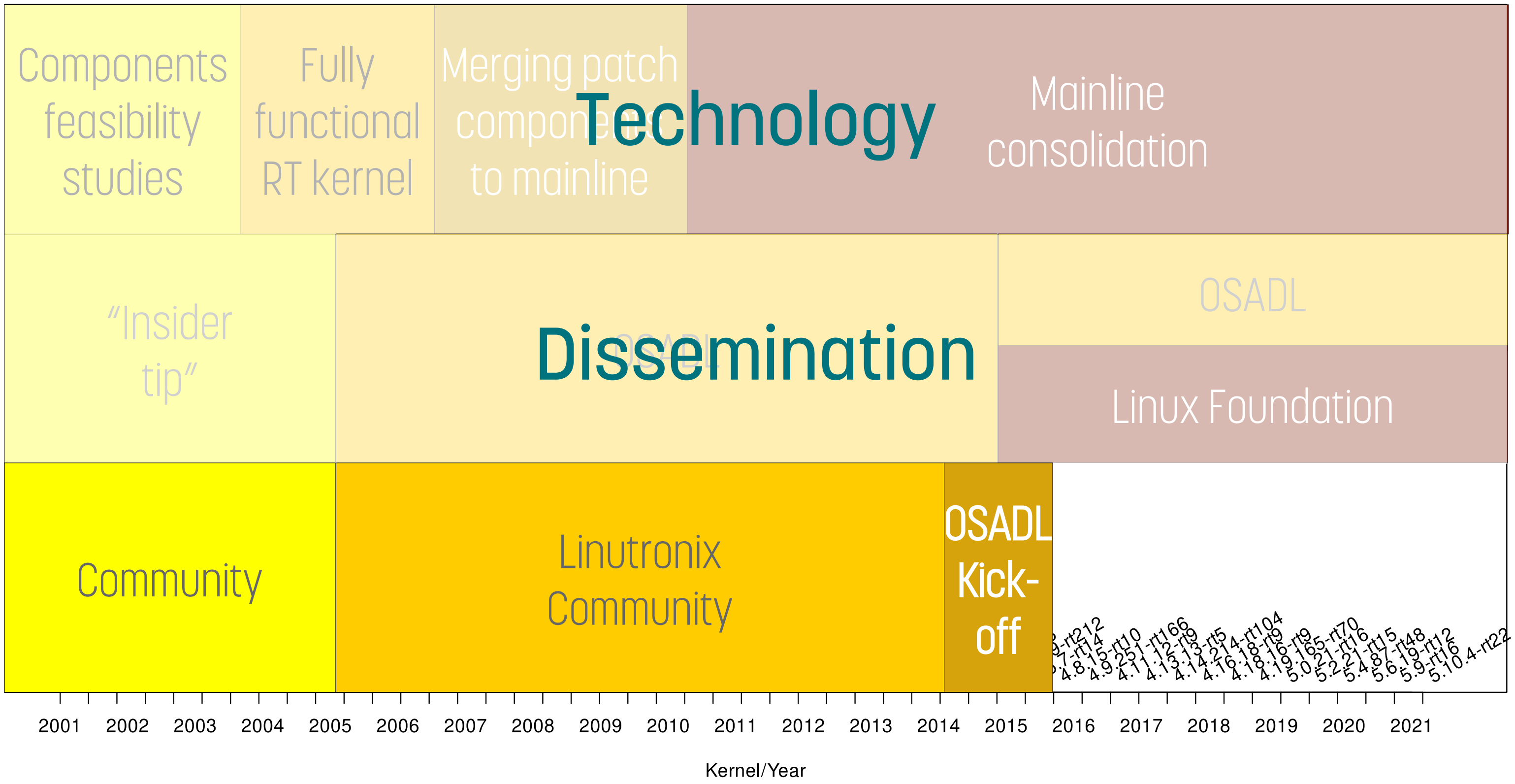
RT_PREEMPT from 2001 to today: Funding



RT_PREEMPT from 2001 to today: Funding



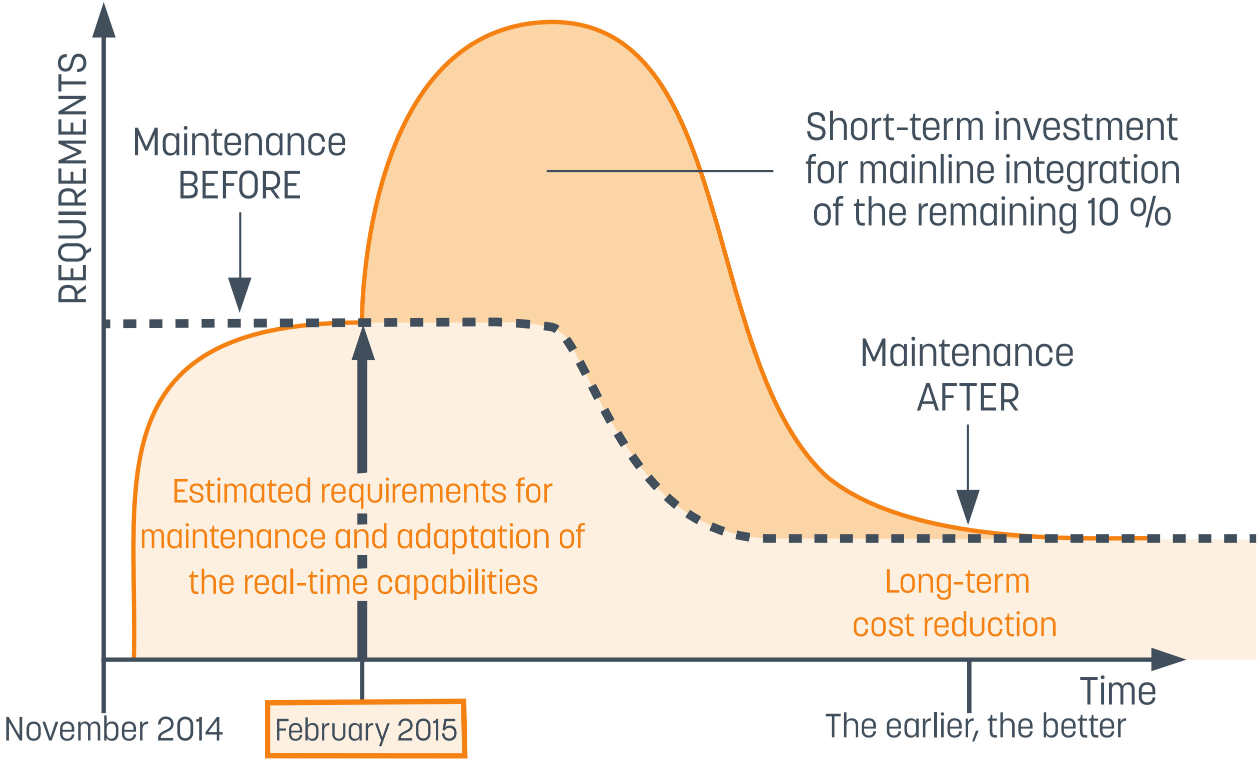
RT_PREEMPT from 2001 to today: Funding



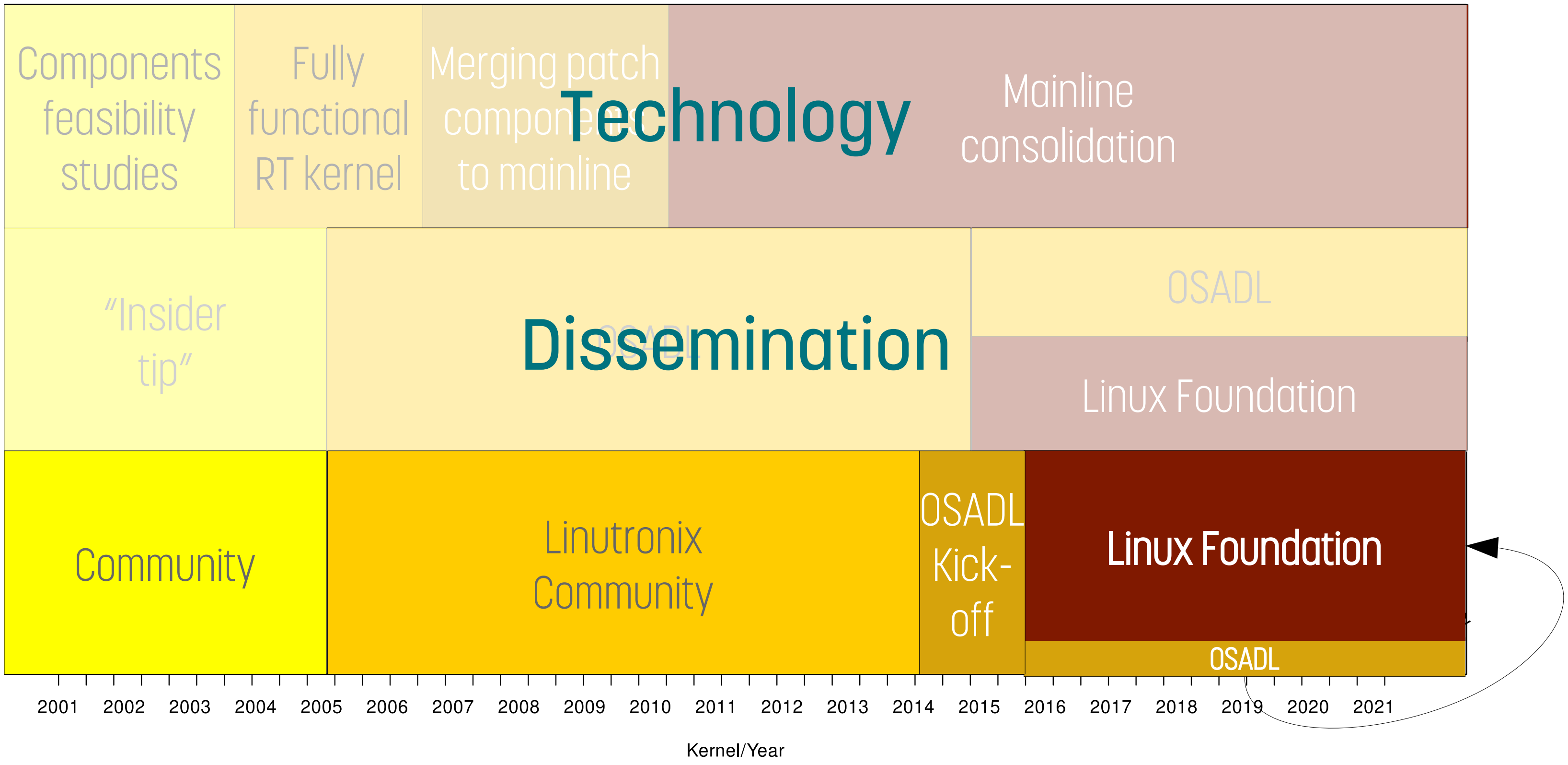
Linux real-time on its way to mainline

Basic lecture: Historical overview about the various steps and components ...

OSADL Kick-off funding



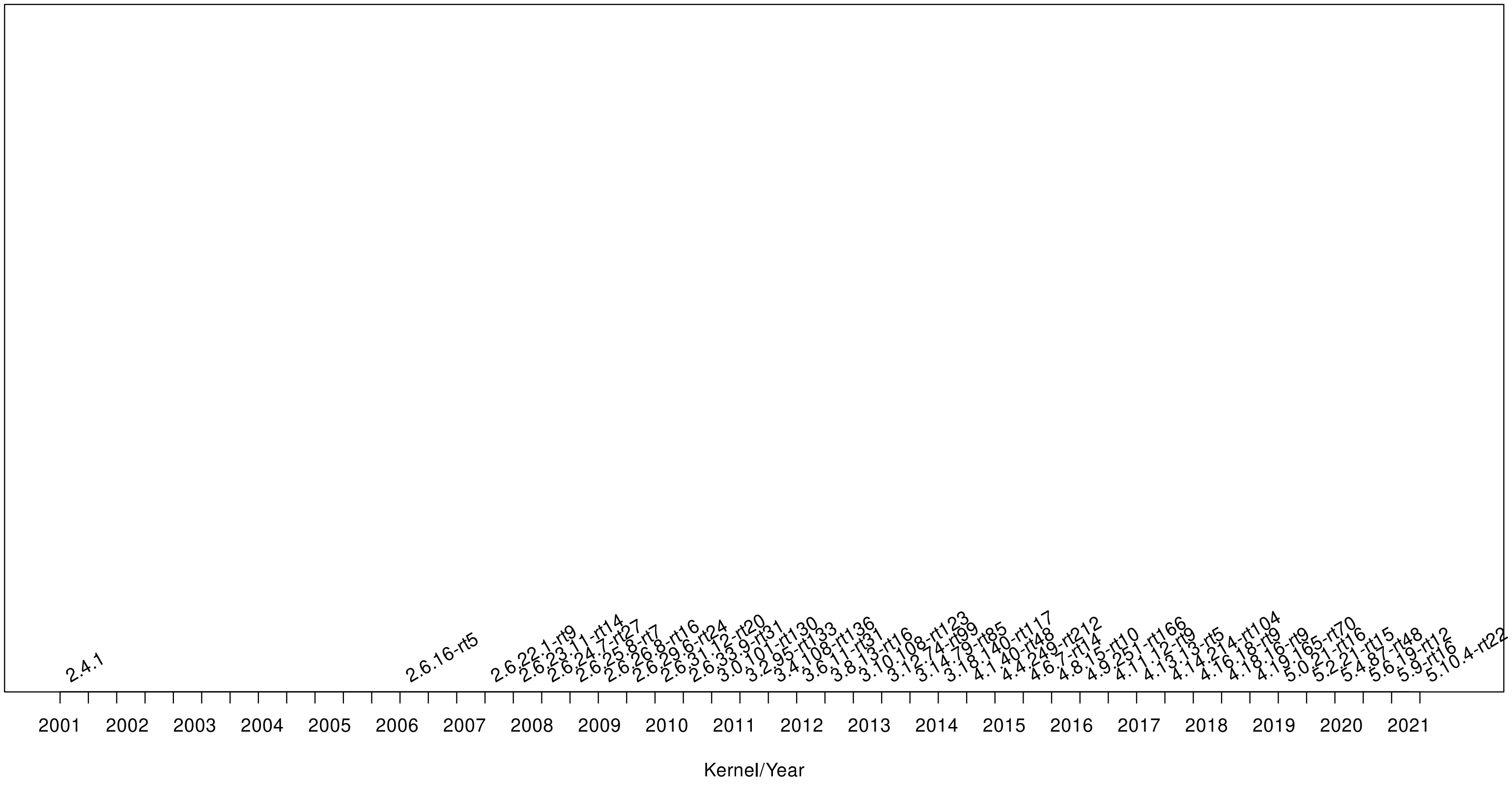
RT_PREEMPT from 2001 to today: Funding



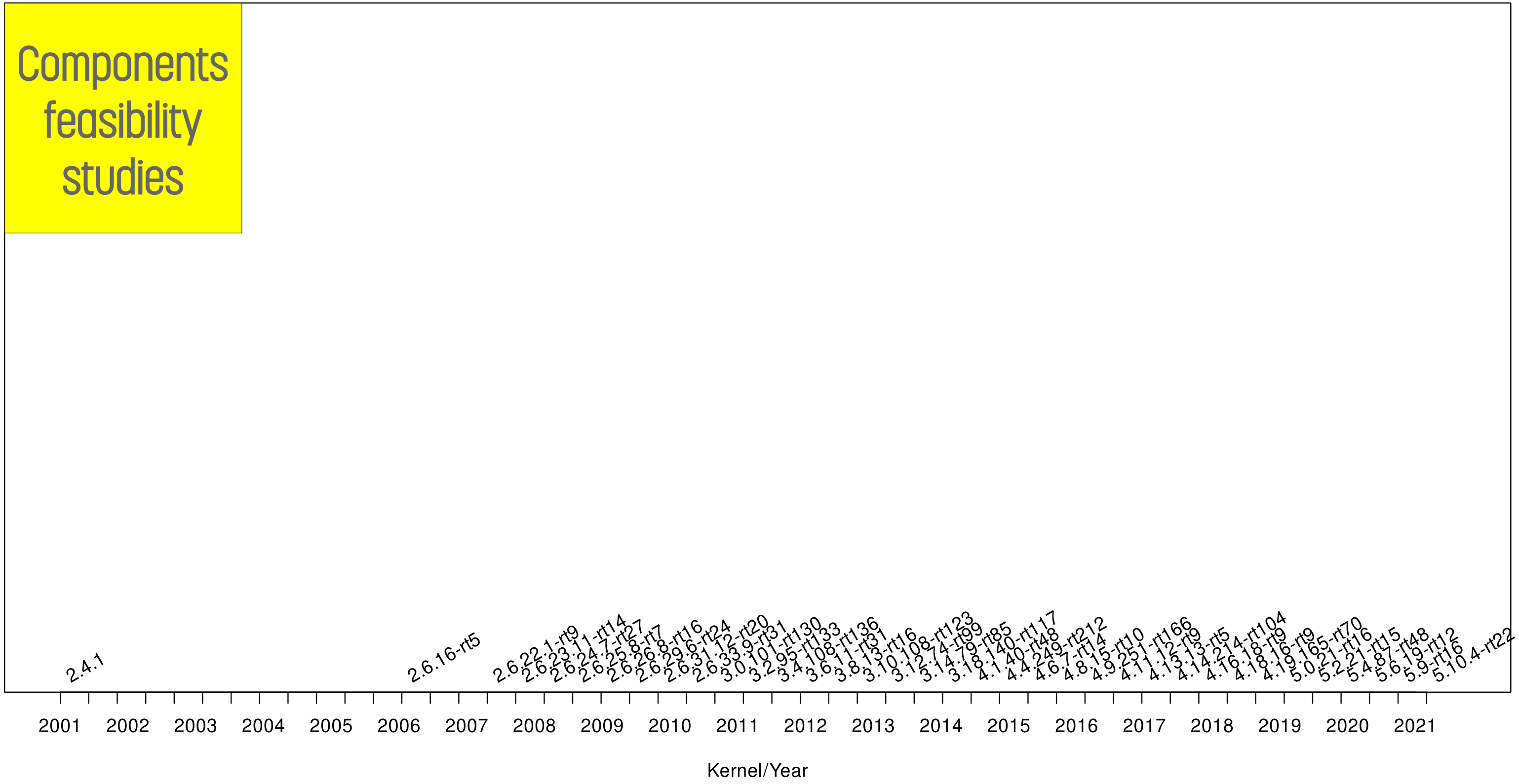
Linux Foundation RTL Collaborative Project

The Real Time Linux collaborative project was established to help coordinate the efforts around mainlining Preempt RT and ensuring that the maintainers have the ability to continue development work, long-term support and future research of RT. In coordination with the broader community, the workgroup aims to encourage broader adoption of RT, improve testing automation and documentation and better prioritize the development roadmap.

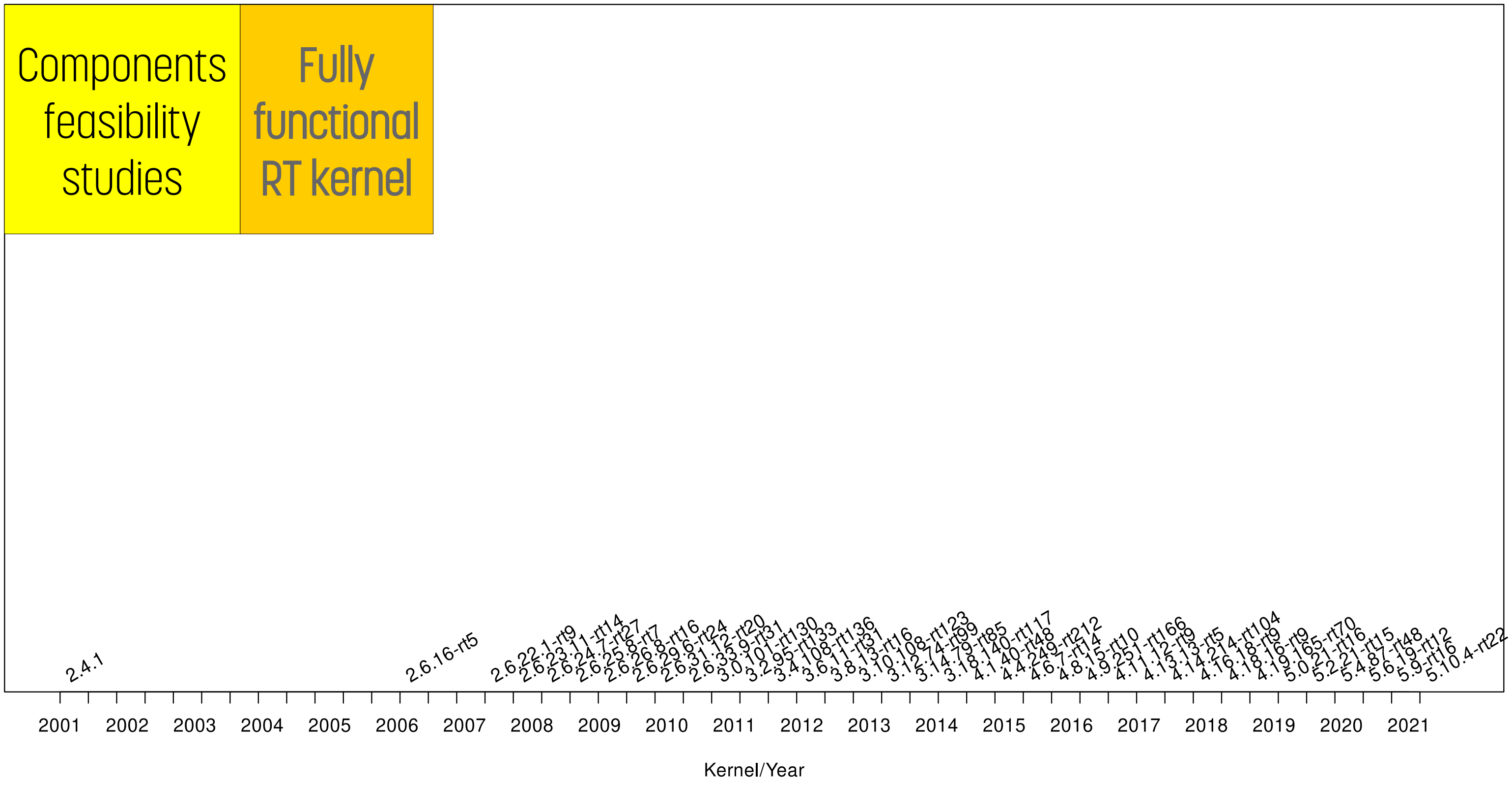
RT_PREEMPT from 2001 to today: The overall puzzle



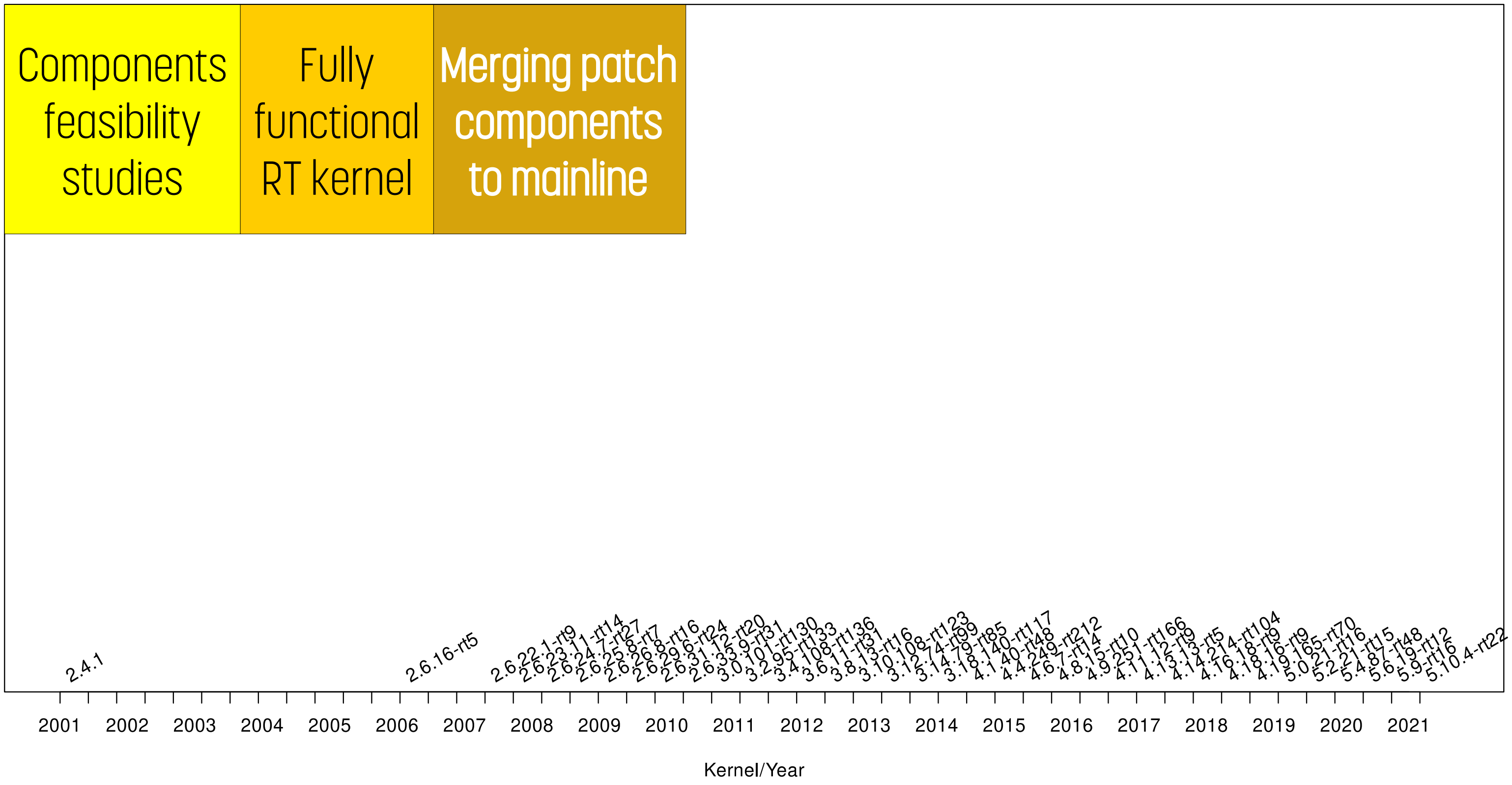
RT_PREEMPT from 2001 to today: The overall puzzle



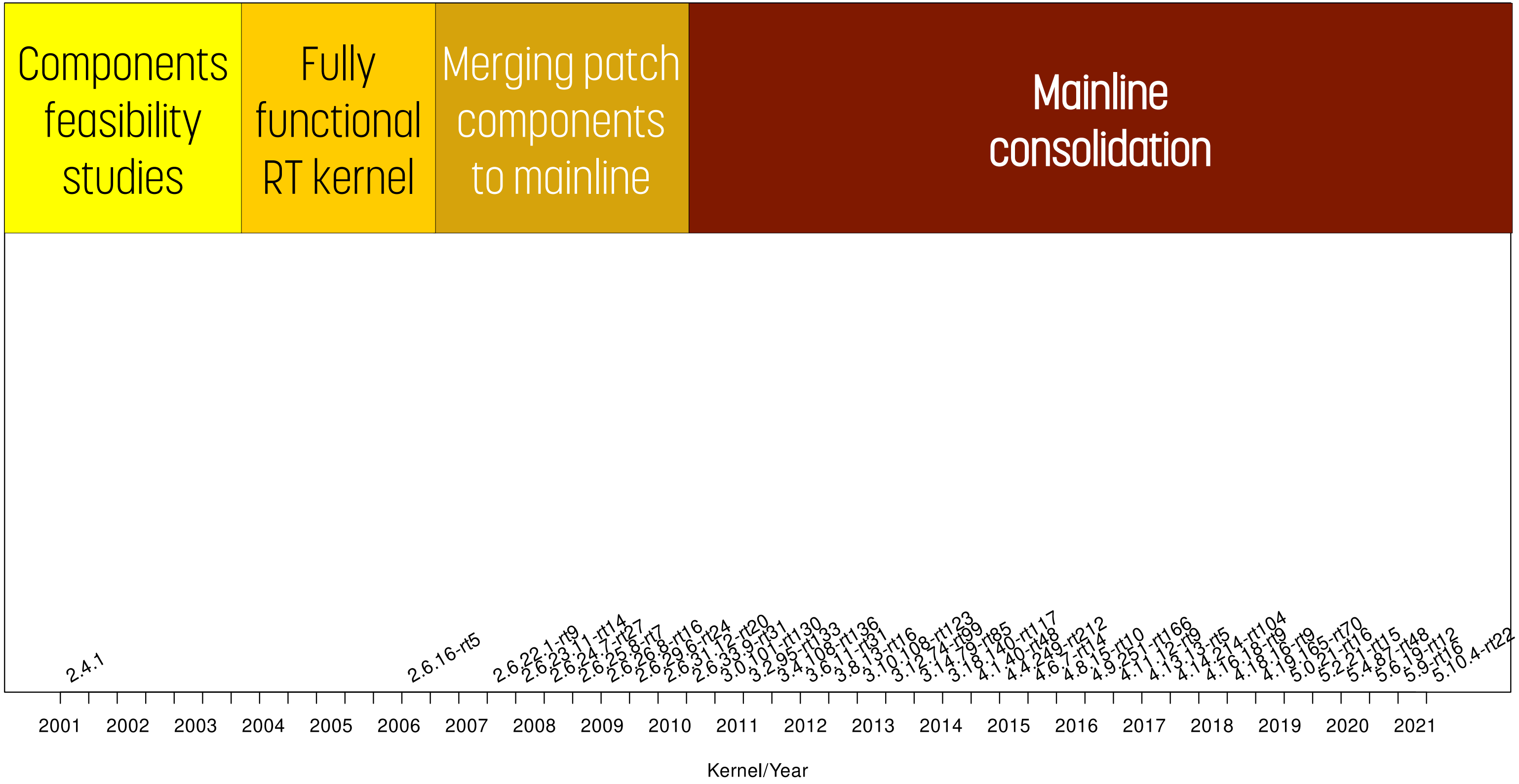
RT_PREEMPT from 2001 to today: The overall puzzle



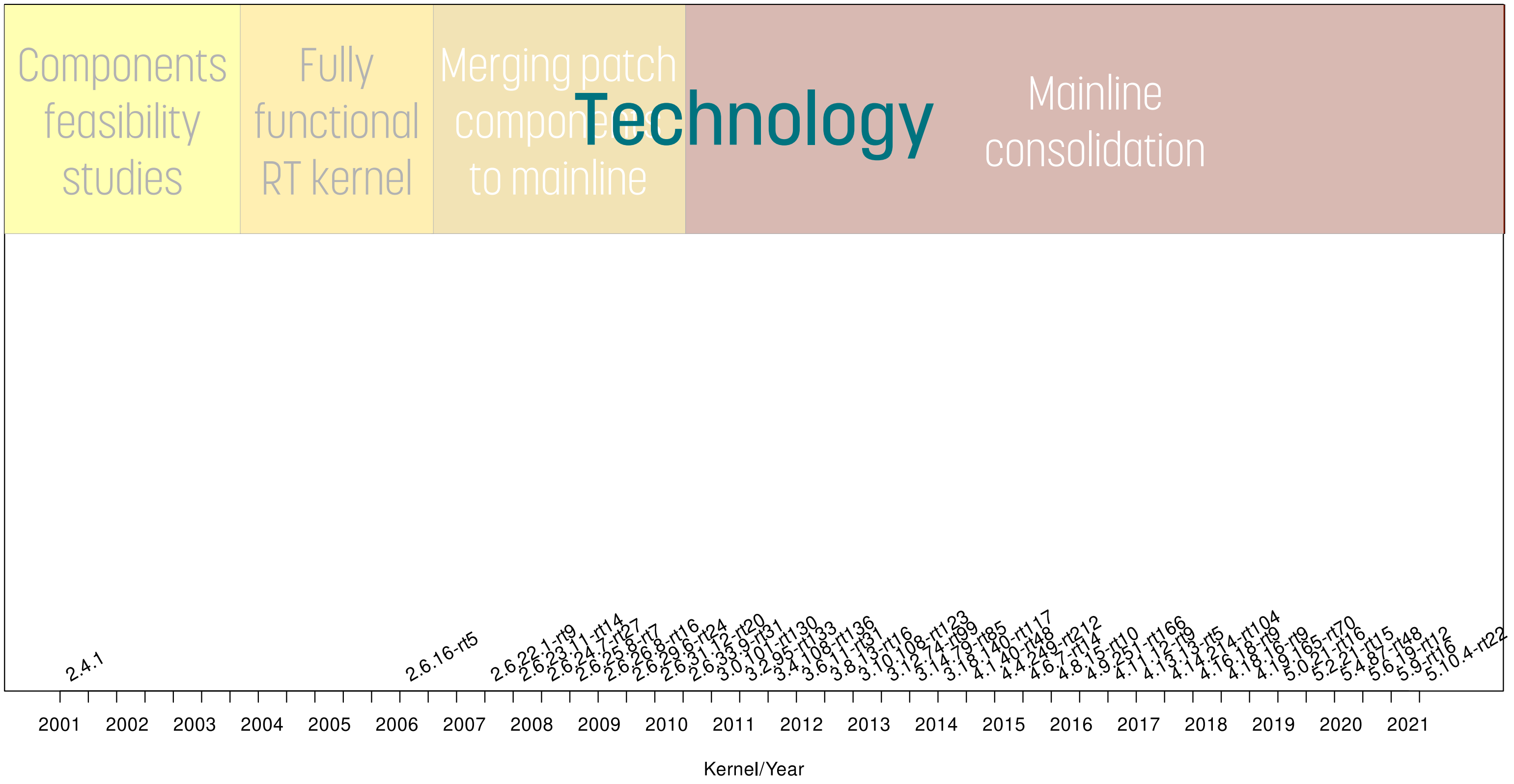
RT_PREEMPT from 2001 to today: The overall puzzle



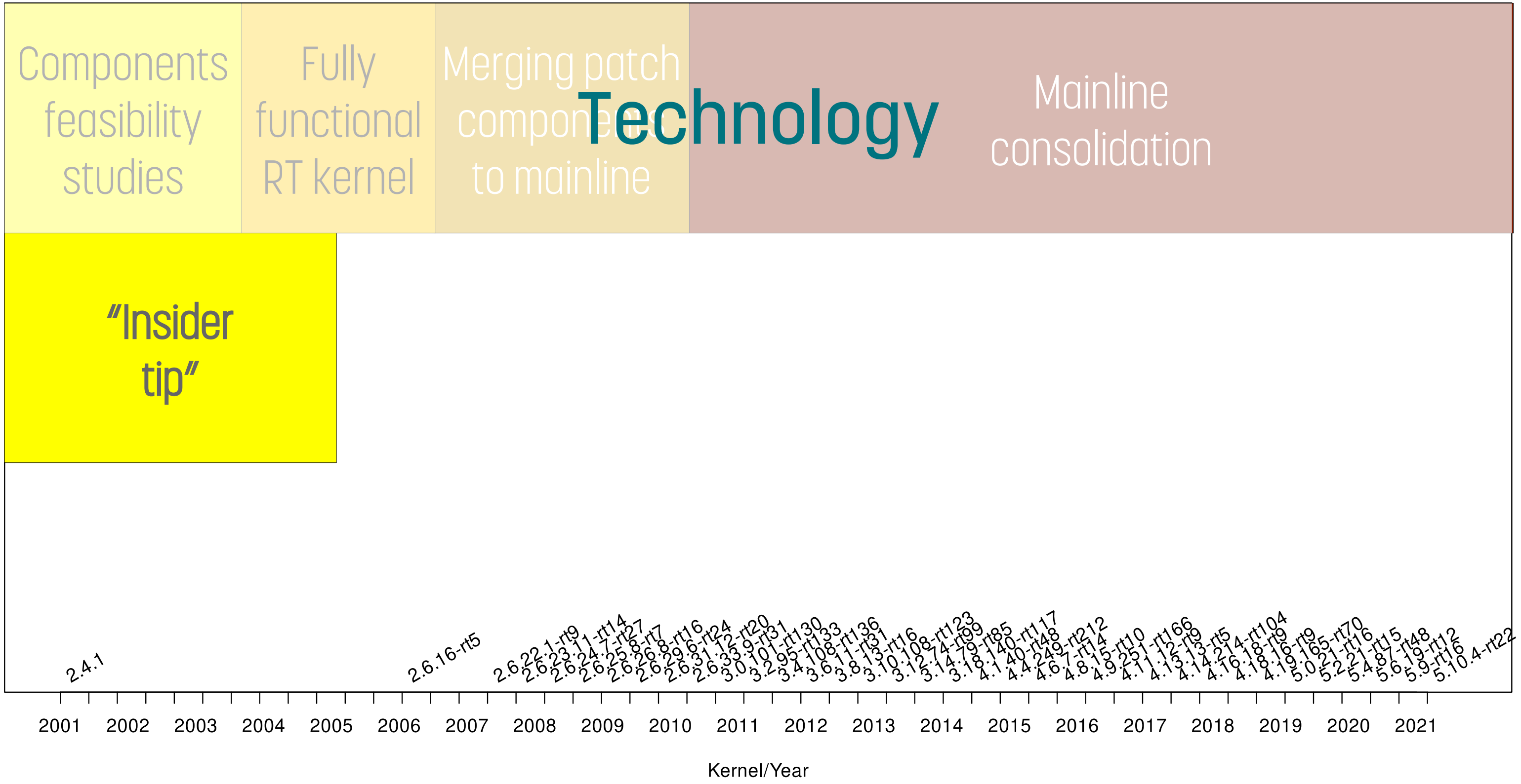
RT_PREEMPT from 2001 to today: The overall puzzle



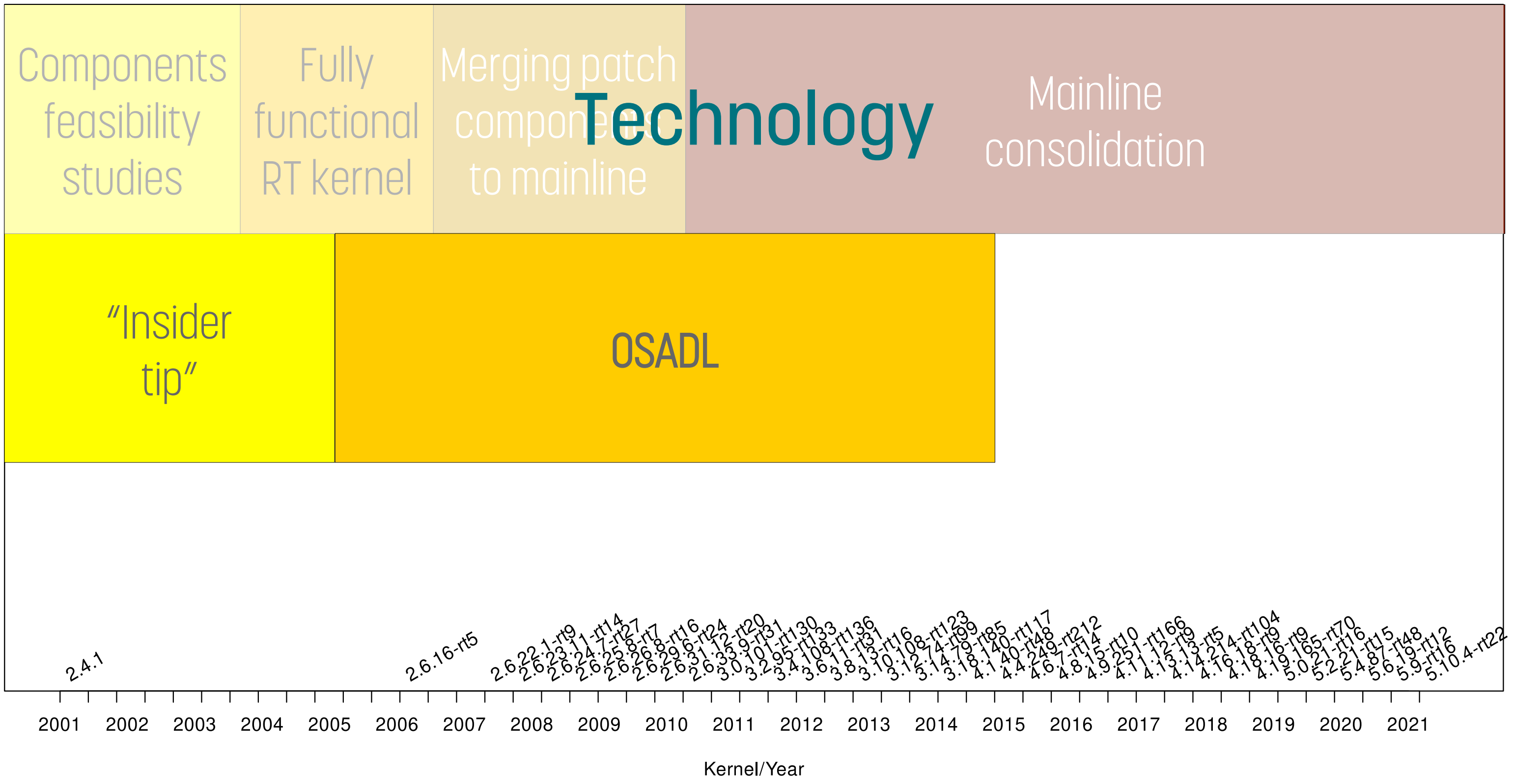
RT_PREEMPT from 2001 to today: The overall puzzle



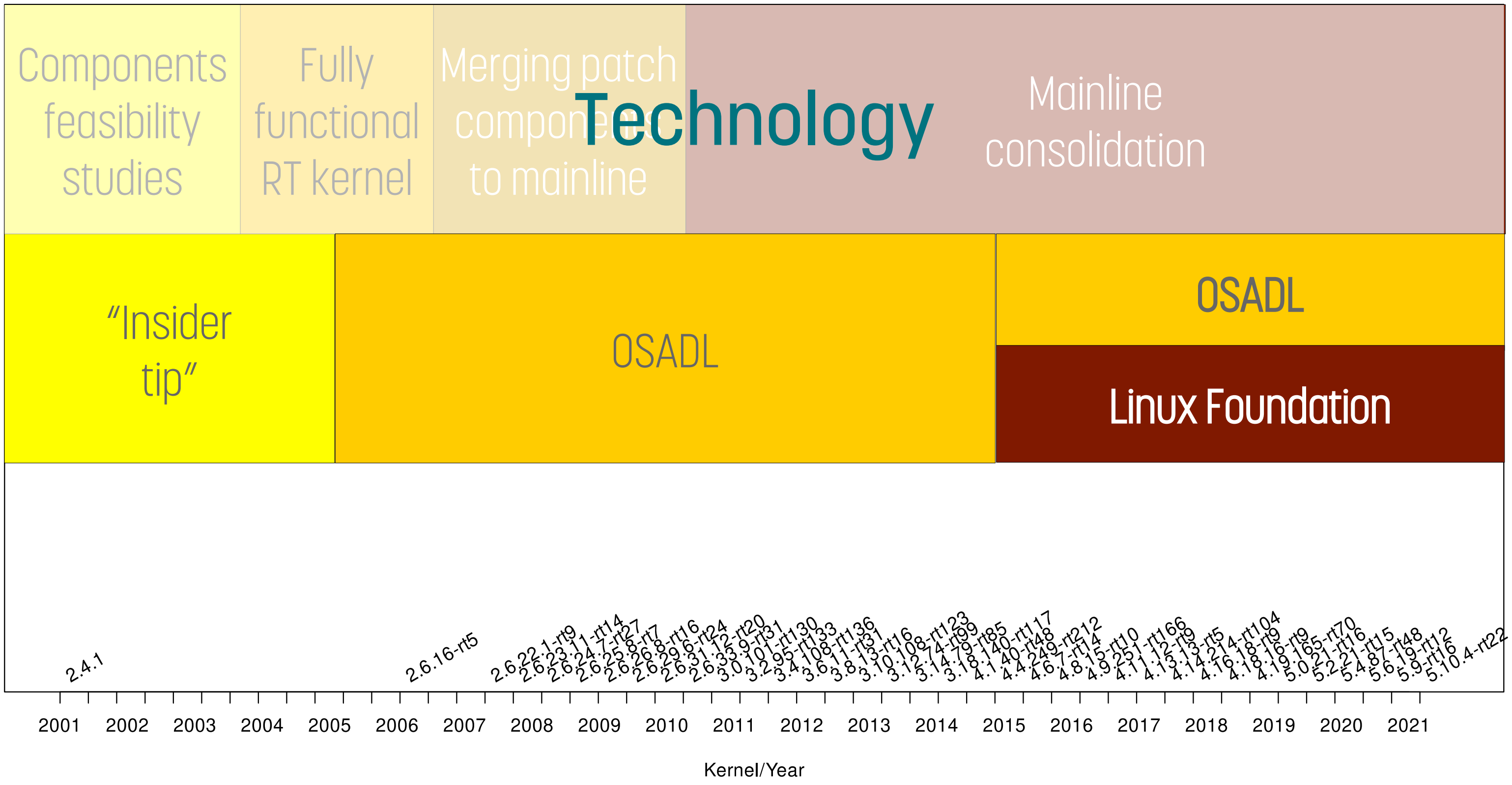
RT_PREEMPT from 2001 to today: The overall puzzle



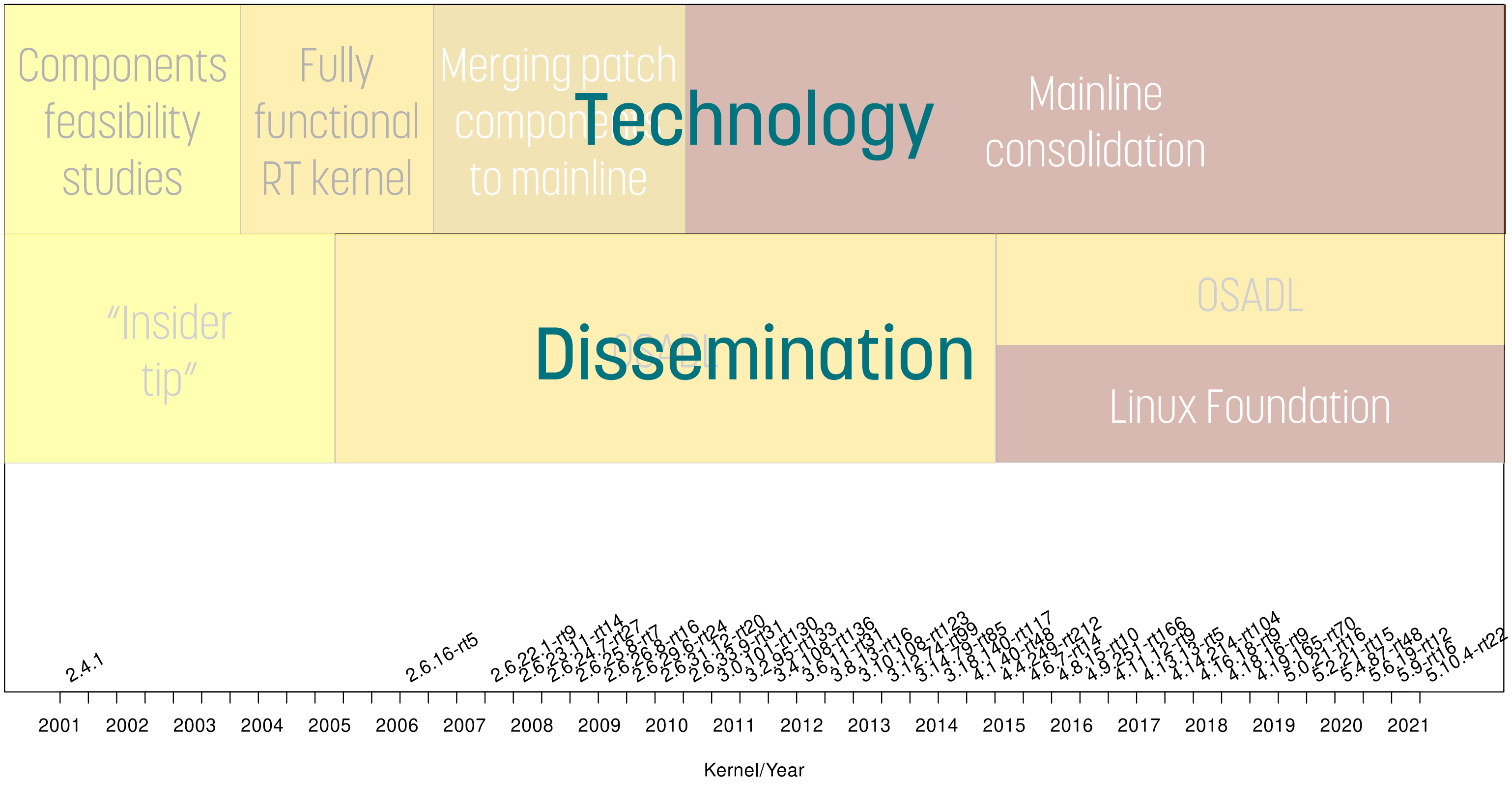
RT_PREEMPT from 2001 to today: The overall puzzle



RT_PREEMPT from 2001 to today: The overall puzzle



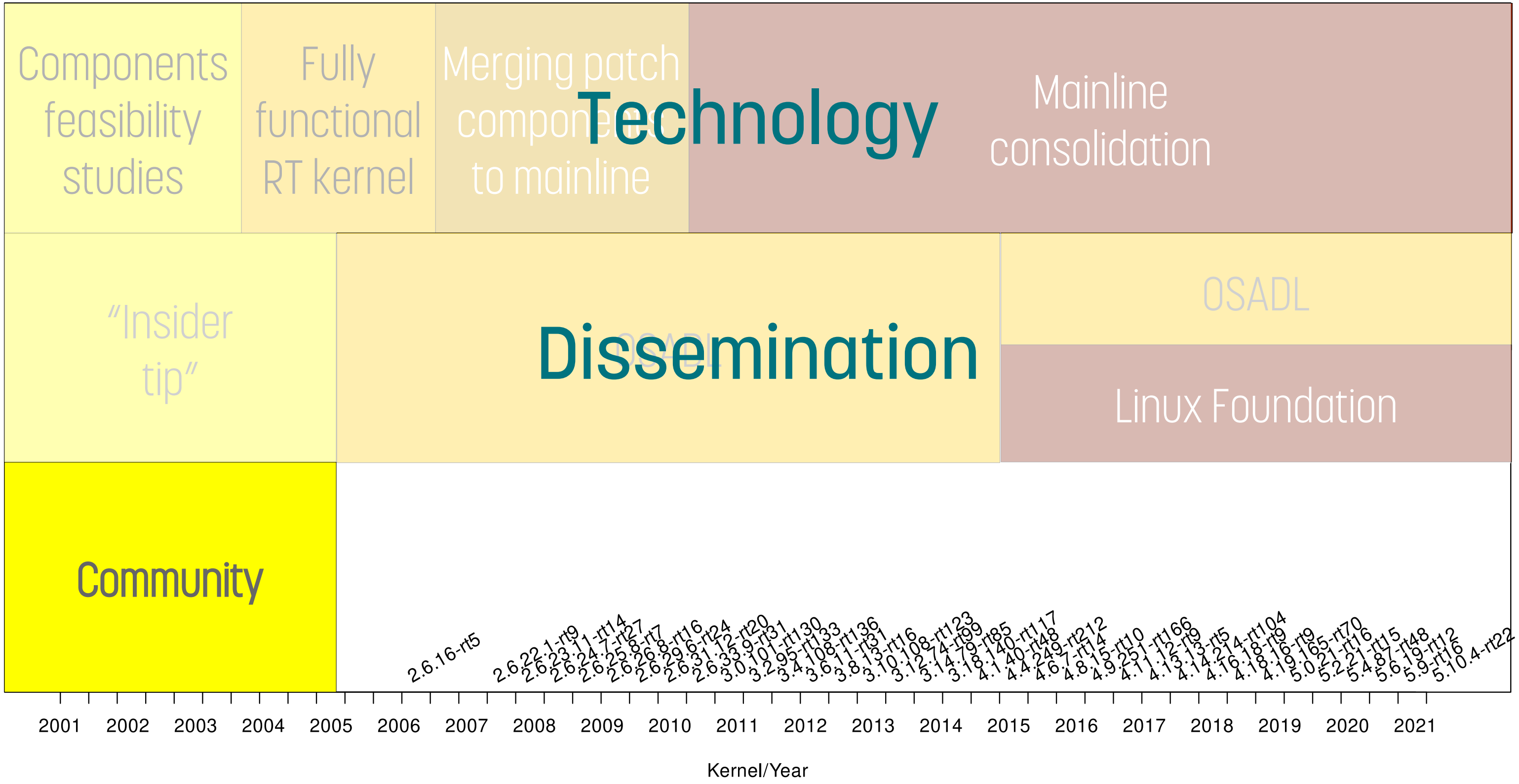
RT_PREEMPT from 2001 to today: The overall puzzle



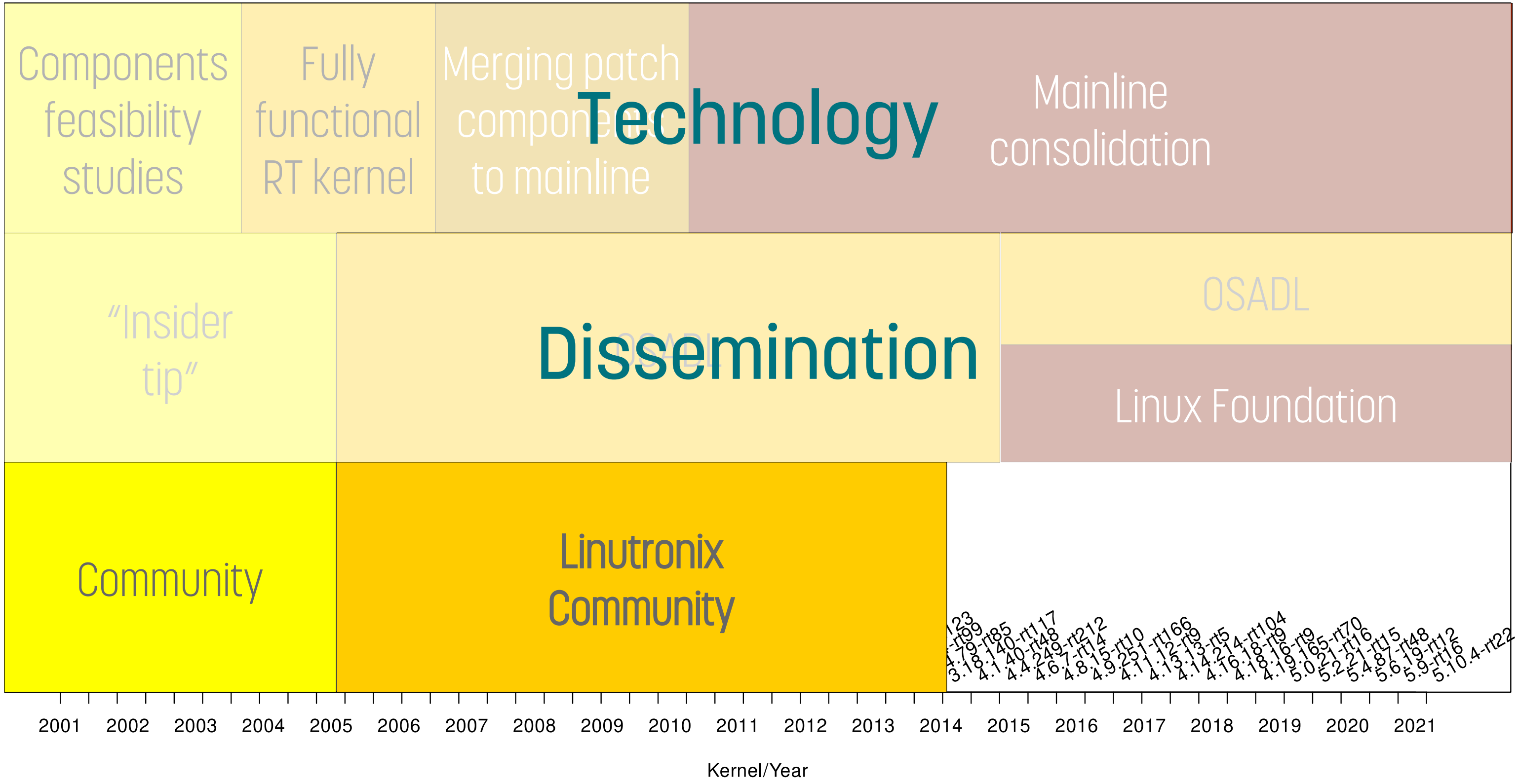
COOL COMPACT OSADL ONLINE LECTURES

OSADL
Open Source Automation Development Lab eG

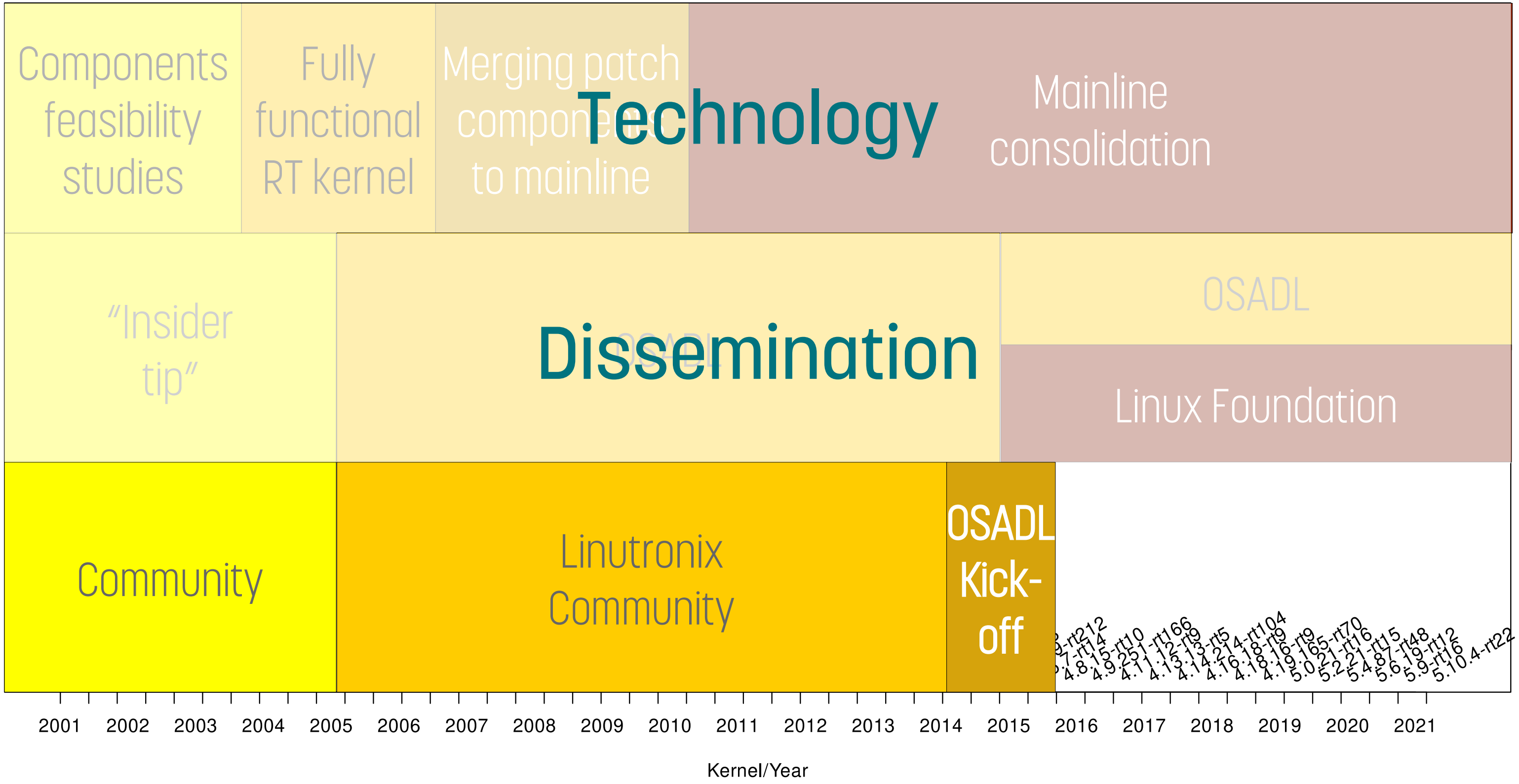
RT_PREEMPT from 2001 to today: The overall puzzle



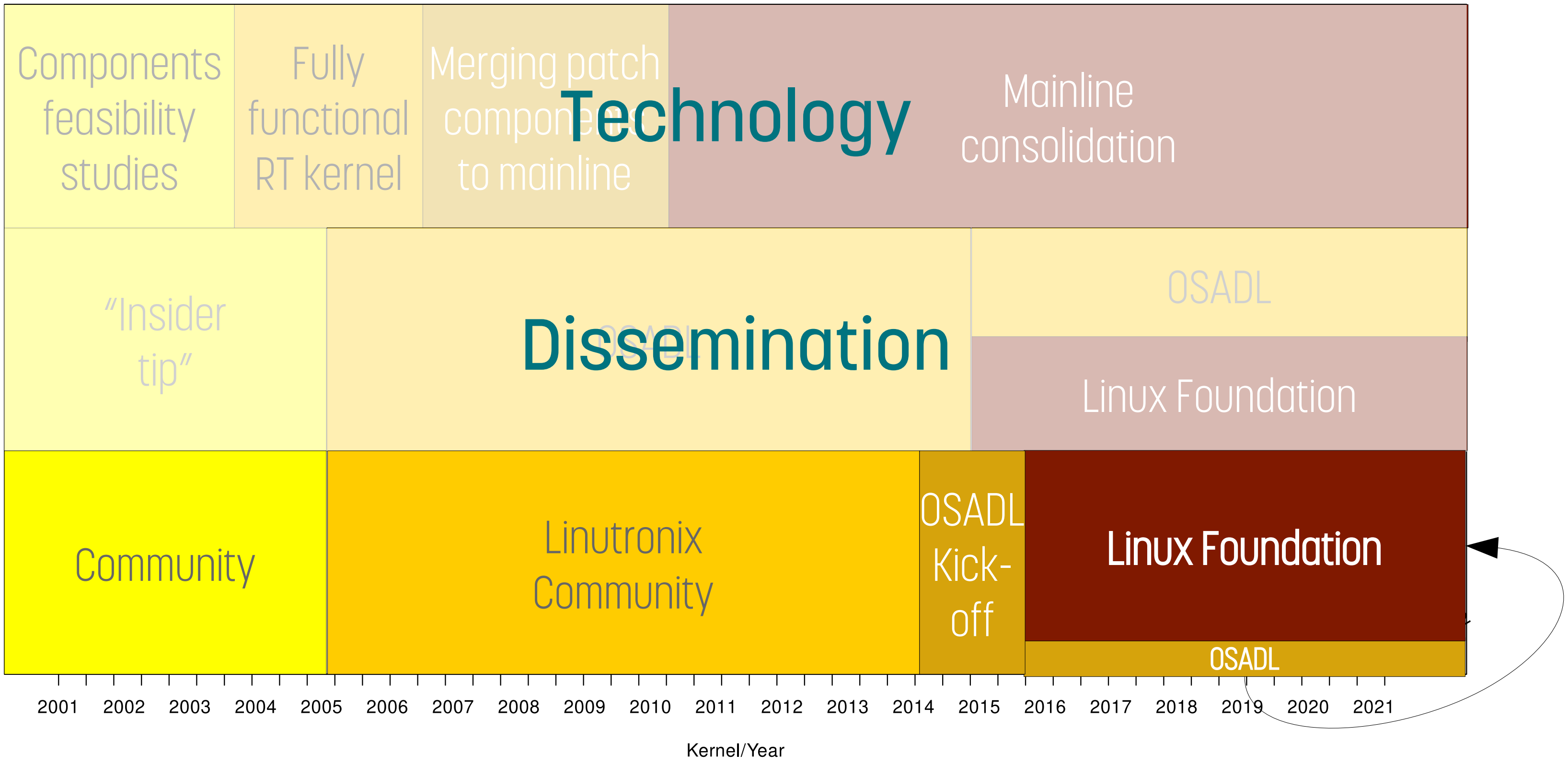
RT_PREEMPT from 2001 to today: The overall puzzle



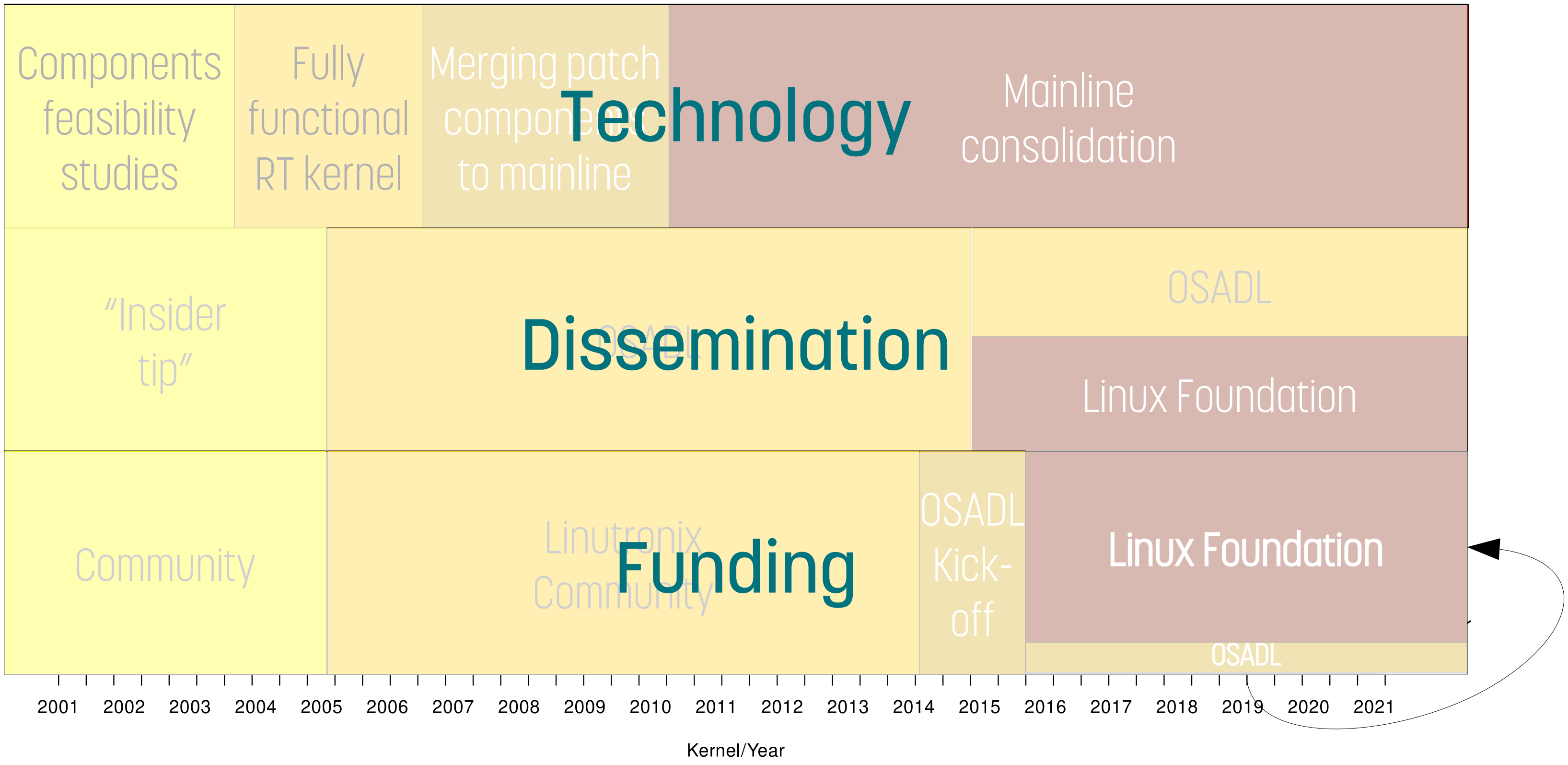
RT_PREEMPT from 2001 to today: The overall puzzle



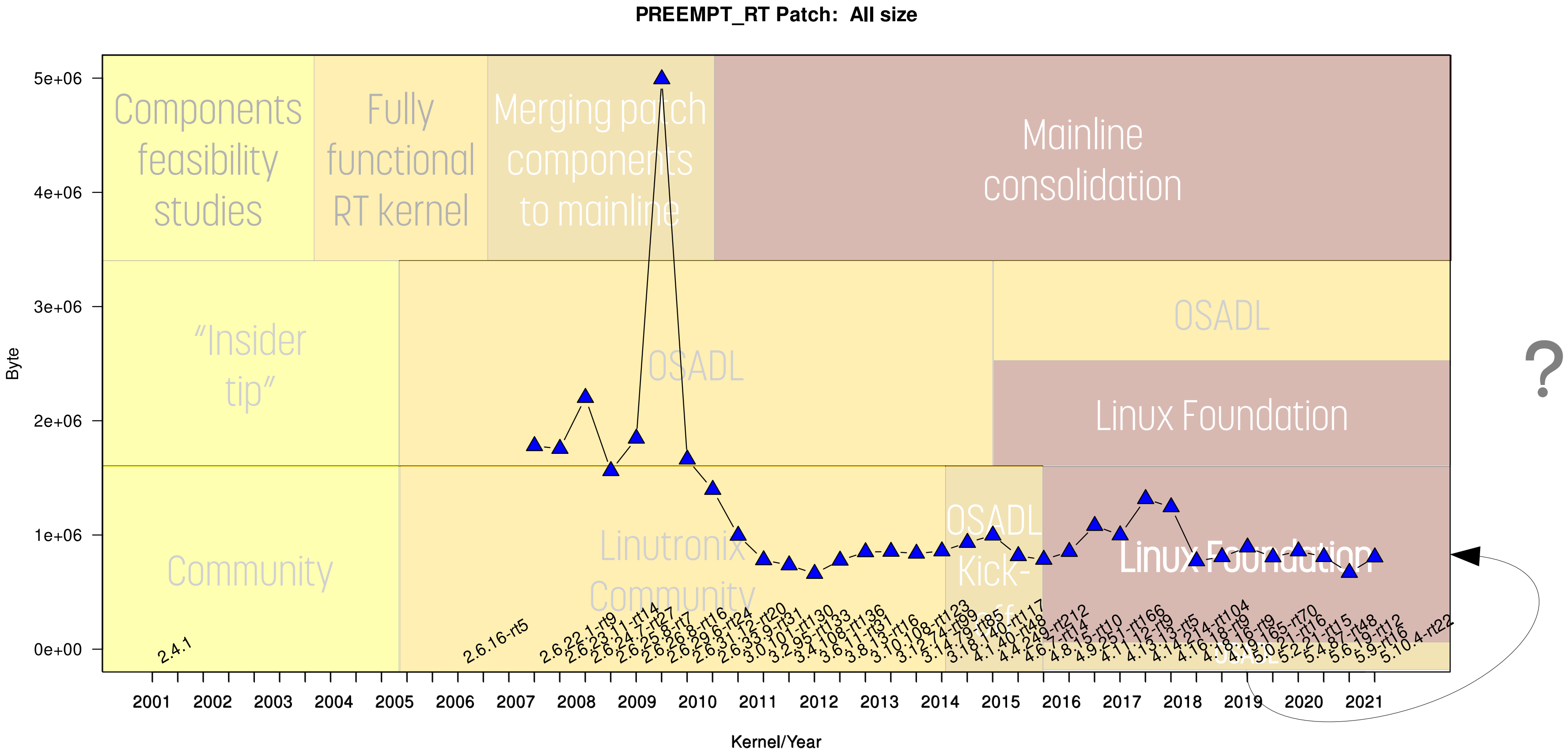
RT_PREEMPT from 2001 to today: The overall puzzle



RT_PREEMPT from 2001 to today: The overall puzzle



RT_PREEMPT from 2001 to today: This is not the end



Linux real-time on its way to mainline

Basic lecture: Historical overview about the various steps and components ...

COOL – Compact OSADL Online Lectures, Wednesday, January 20, 2021