THE CASE FOR AN OPINIONATED, THEORY-ORIENTED REAL-TIME OPERATING SYSTEM

NGOSCPS'19

April 15, 2019

Björn Brandenburg

bbb@mpi-sws.org

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

European Research Council

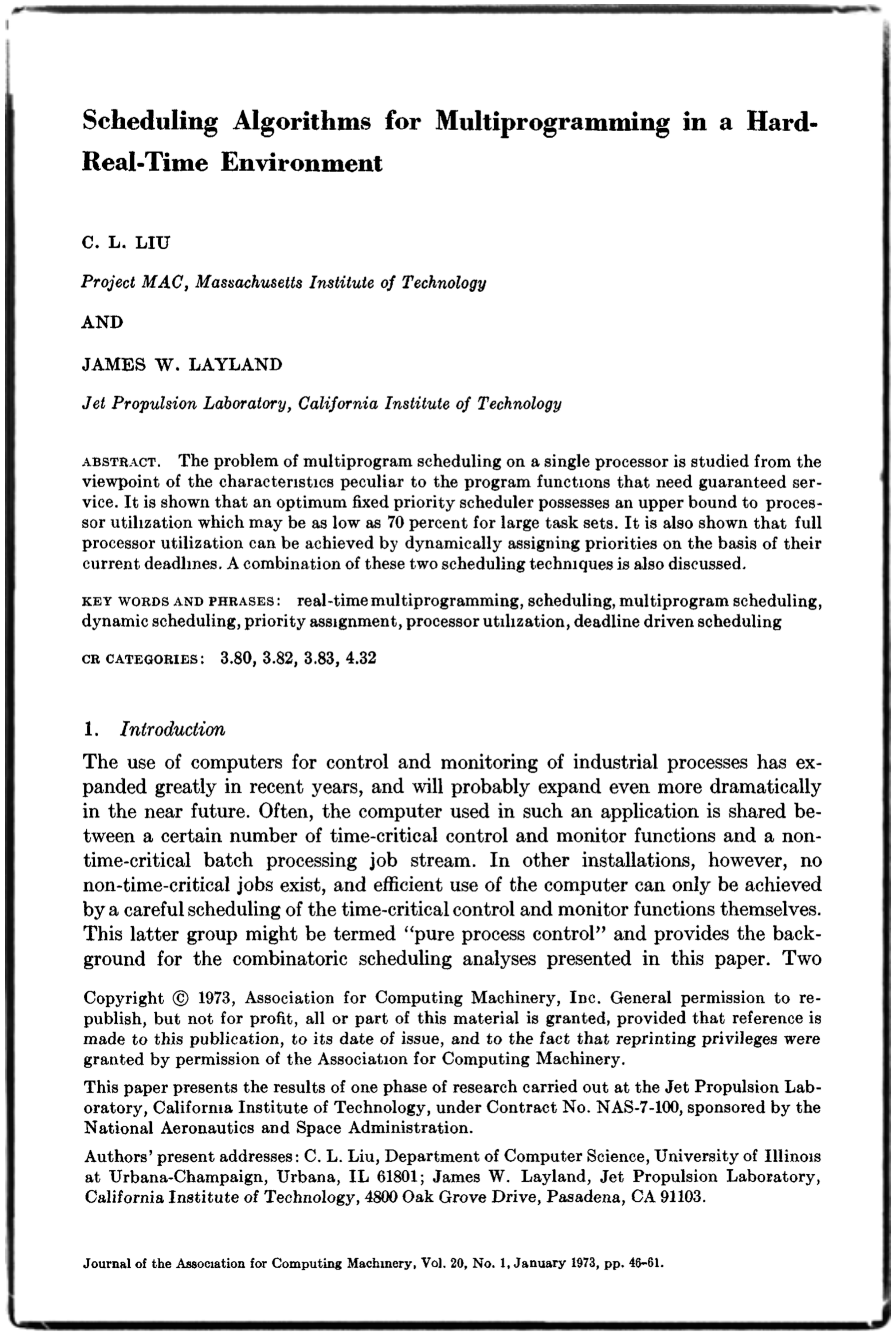
Established by the European Commission

MAX-PLANCK-GESELLSCHAFT

REAL-TIME SYSTEMS

A SUCCESS STORY

FIVE DECADES OF REAL-TIME SYSTEMS RESEARCH



Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

C. L. LIU

Project MAC, Massachusetts Institute of Technology

AND

JAMES W. LAYLAND

Jet Propulsion Laboratory, California Institute of Technology

ABSTRACT. The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor utilization which may be as low as 70 percent for large task sets. It is also shown that full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines. A combination of these two scheduling techniques is also discussed.

KEY WORDS AND PHRASES: real-time multiprogramming, scheduling, multiprogram scheduling, dynamic scheduling, priority assignment, processor utilization, deadline driven scheduling

CR CATEGORIES: 3.80, 3.82, 3.83, 4.32

1. *Introduction*

The use of computers for control and monitoring of industrial processes has expanded greatly in recent years, and will probably expand even more dramatically in the near future. Often, the computer used in such an application is shared between a certain number of time-critical control and monitor functions and a non-time-critical batch processing job stream. In other installations, however, no non-time-critical jobs exist, and efficient use of the computer can only be achieved by a careful scheduling of the time-critical control and monitor functions themselves. This latter group might be termed "pure process control" and provides the background for the combinatoric scheduling analyses presented in this paper. Two

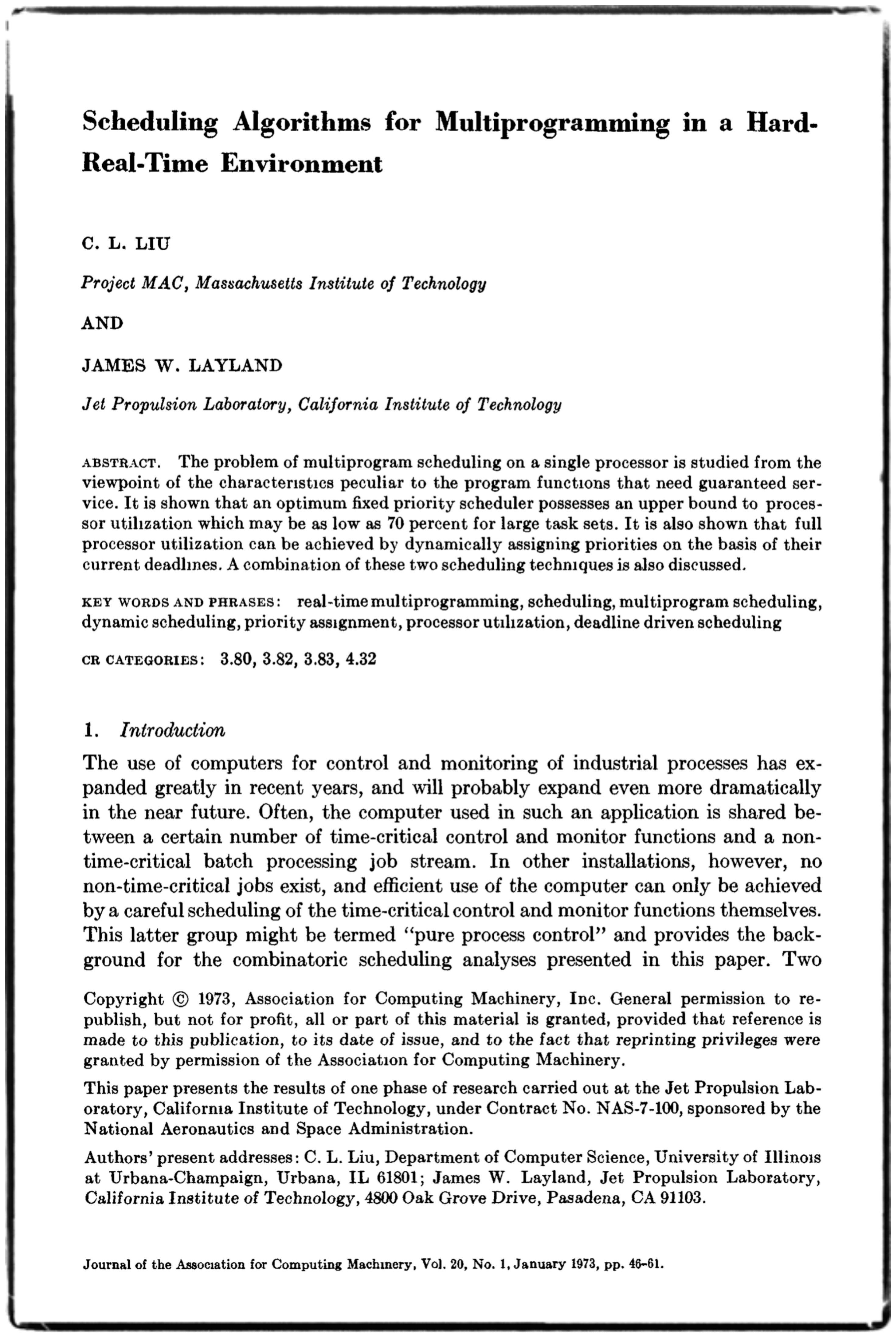
This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS-7-100, sponsored by the National Aeronautics and Space Administration.

Authors' present addresses: C. L. Liu, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801; James W. Layland, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91103.

Journal of the Association for Computing Machinery, Vol. 20, No. 1, January 1973, pp. 46-61.

(Liu & Layland, 1973)

FIVE DECADES OF REAL-TIME SYSTEMS RESEARCH



Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

C. L. LIU

Project MAC, Massachusetts Institute of Technology

AND

JAMES W. LAYLAND

Jet Propulsion Laboratory, California Institute of Technology

ABSTRACT. The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor utilization which may be as low as 70 percent for large task sets. It is also shown that full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines. A combination of these two scheduling techniques is also discussed.

KEY WORDS AND PHRASES: real-time multiprogramming, scheduling, multiprogram scheduling, dynamic scheduling, priority assignment, processor utilization, deadline driven scheduling

CR CATEGORIES: 3.80, 3.82, 3.83, 4.32

1. *Introduction*

The use of computers for control and monitoring of industrial processes has expanded greatly in recent years, and will probably expand even more dramatically in the near future. Often, the computer used in such an application is shared between a certain number of time-critical control and monitor functions and a non-time-critical batch processing job stream. In other installations, however, no non-time-critical jobs exist, and efficient use of the computer can only be achieved by a careful scheduling of the time-critical control and monitor functions themselves. This latter group might be termed "pure process control" and provides the background for the combinatoric scheduling analyses presented in this paper. Two

This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS-7-100, sponsored by the National Aeronautics and Space Administration.

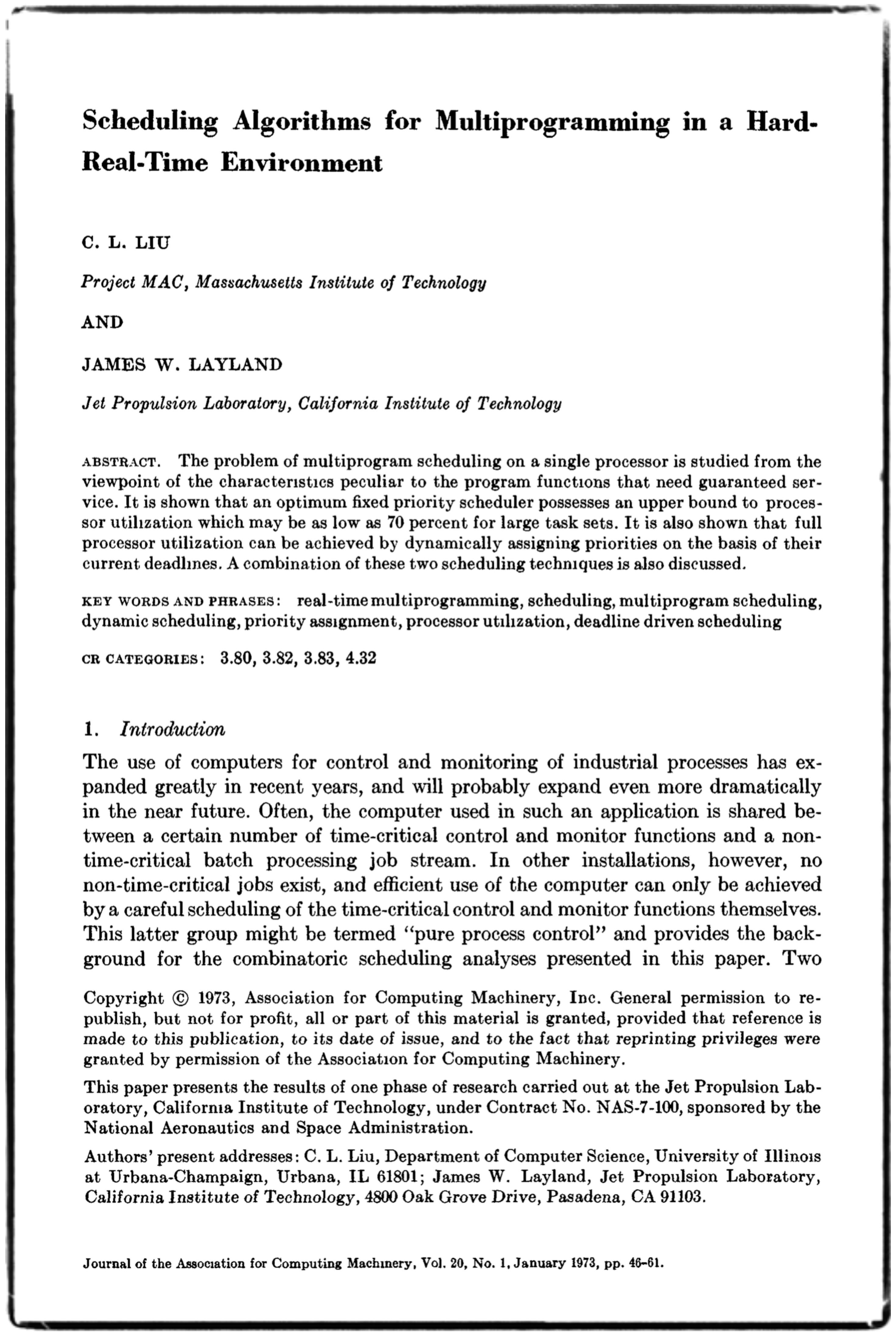
Authors' present addresses: C. L. Liu, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801; James W. Layland, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91103.

Journal of the Association for Computing Machinery, Vol. 20, No. 1, January 1973, pp. 46-61.

(Liu & Layland, 1973)

≈**12k citations** (Google Scholar)

FIVE DECADES OF REAL-TIME SYSTEMS RESEARCH



Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment

C. L. LIU

Project MAC, Massachusetts Institute of Technology

AND

JAMES W. LAYLAND

Jet Propulsion Laboratory, California Institute of Technology

ABSTRACT. The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to processor utilization which may be as low as 70 percent for large task sets. It is also shown that full processor utilization can be achieved by dynamically assigning priorities on the basis of their current deadlines. A combination of these two scheduling techniques is also discussed.

KEY WORDS AND PHRASES: real-time multiprogramming, scheduling, multiprogram scheduling, dynamic scheduling, priority assignment, processor utilization, deadline driven scheduling

CR CATEGORIES: 3.80, 3.82, 3.83, 4.32

1. *Introduction*

The use of computers for control and monitoring of industrial processes has expanded greatly in recent years, and will probably expand even more dramatically in the near future. Often, the computer used in such an application is shared between a certain number of time-critical control and monitor functions and a non-time-critical batch processing job stream. In other installations, however, no non-time-critical jobs exist, and efficient use of the computer can only be achieved by a careful scheduling of the time-critical control and monitor functions themselves. This latter group might be termed "pure process control" and provides the background for the combinatoric scheduling analyses presented in this paper. Two

Copyright © 1973, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that reference is made to this publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS-7-100, sponsored by the National Aeronautics and Space Administration.

Authors' present addresses: C. L. Liu, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801; James W. Layland, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91103.

Journal of the Association for Computing Machinery, Vol. 20, No. 1, January 1973, pp. 46–61.

(Liu & Layland, 1973)

≈12k citations (Google Scholar)

➔ *(where) does this get used in practice???*

MAJOR USERS OF REAL-TIME SYSTEMS TECHNOLOGY

MAJOR USERS OF REAL-TIME SYSTEMS TECHNOLOGY

BOEING
AIRBUS
BOSCH
THALES
DENSO
Crafting the Core
Honeywell
ABB
Rockwell Collins

To name just a few examples…

MAJOR USERS OF REAL-TIME SYSTEMS TECHNOLOGY

Large international companies with
large R&D budgets and **dedicated real-time specialists**

➔ *Tremendous amount of in-house real-time expertise!*

MAJOR USERS OF REAL-TIME SYSTEMS TECHNOLOGY

Define "use"...

...formal, sound timing analysis?

Large international companies with

large R&D budgets and **dedicated real-time specialists**

➔ *Tremendous amount of in-house real-time expertise!*

BUT WHAT ABOUT USERS IN THE "LONG TAIL"?

much expertise → little expertise

BUT WHAT ABOUT USERS IN THE "LONG TAIL"?

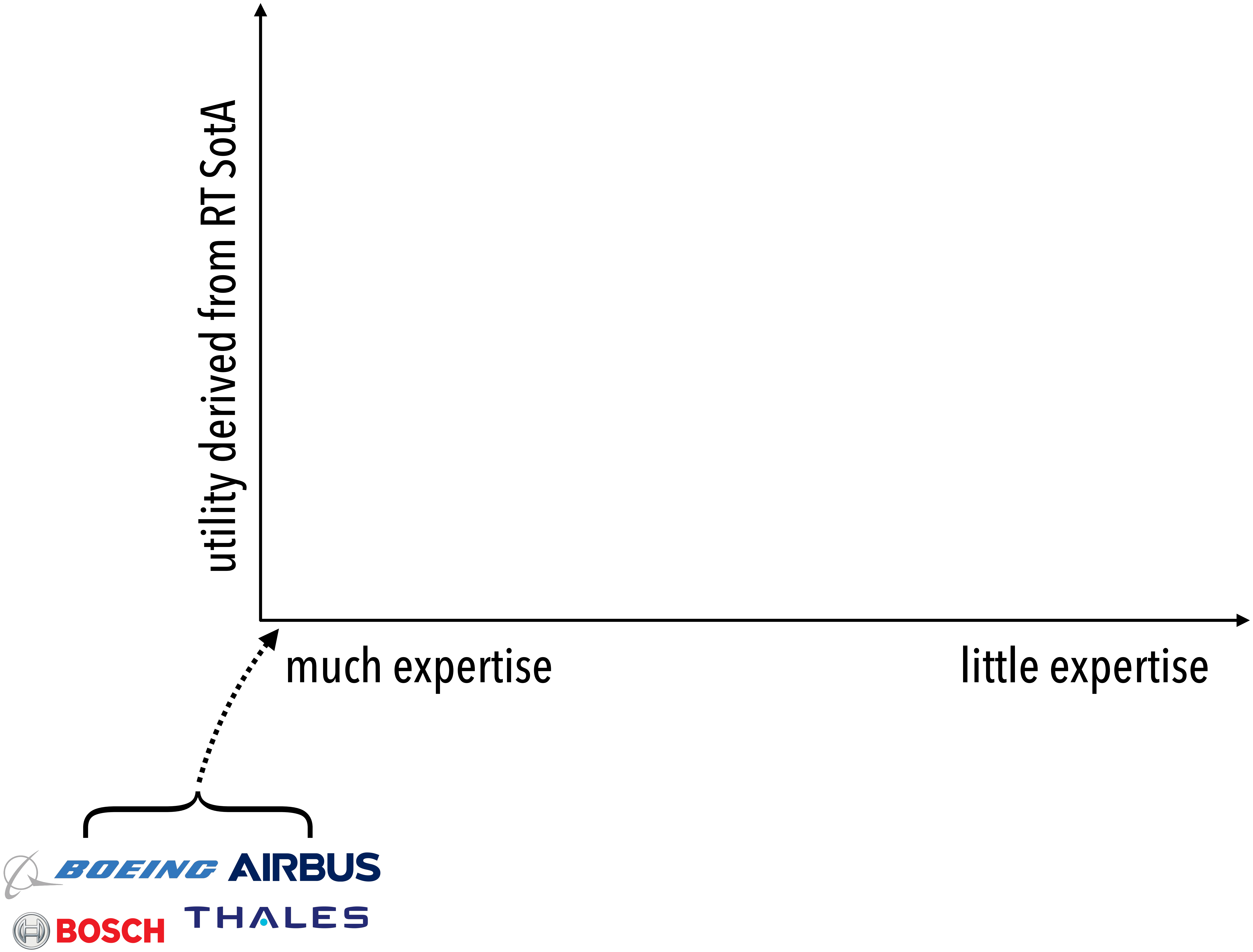
much expertise

little expertise

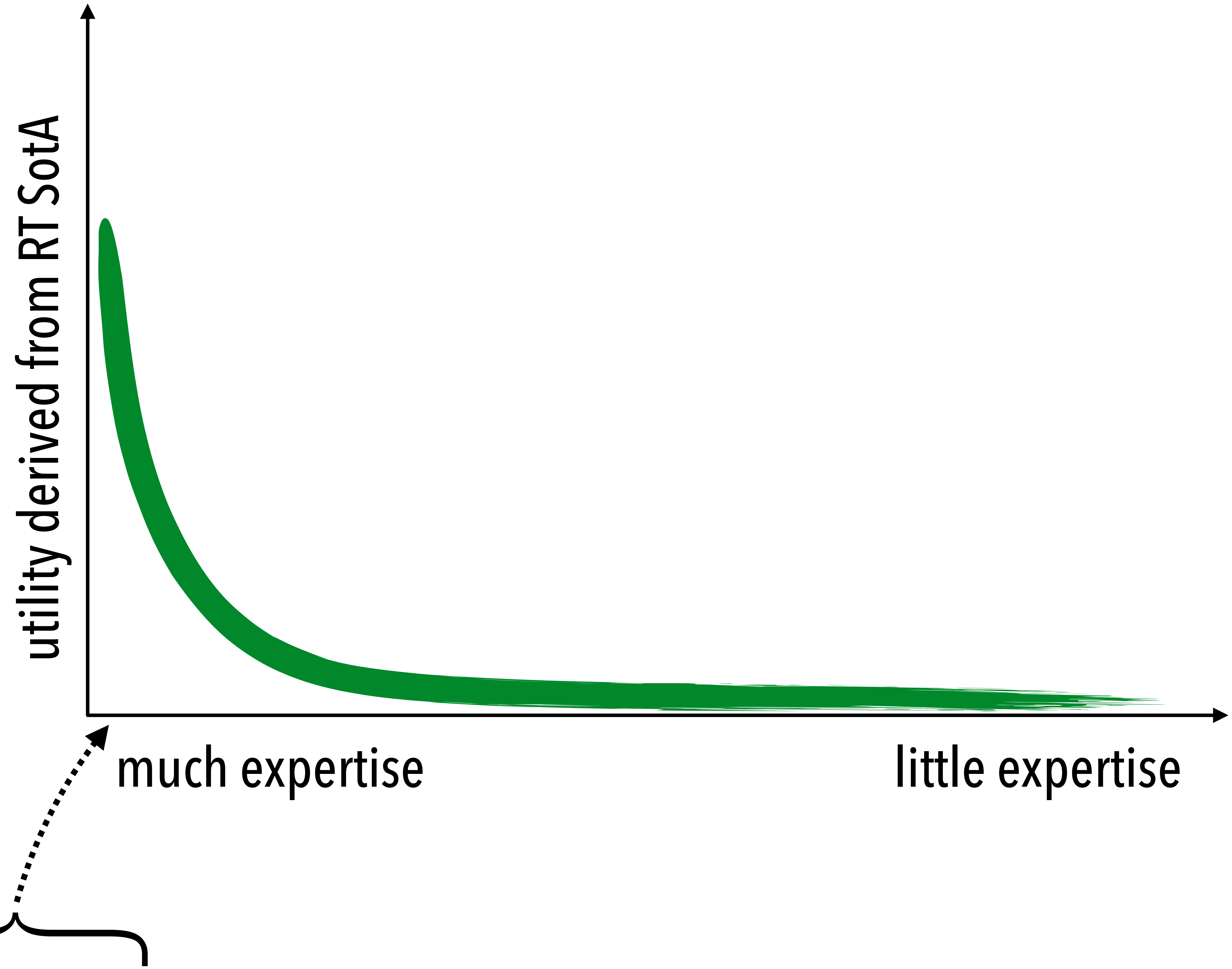
BOEING AIRBUS

BOSCH THALES

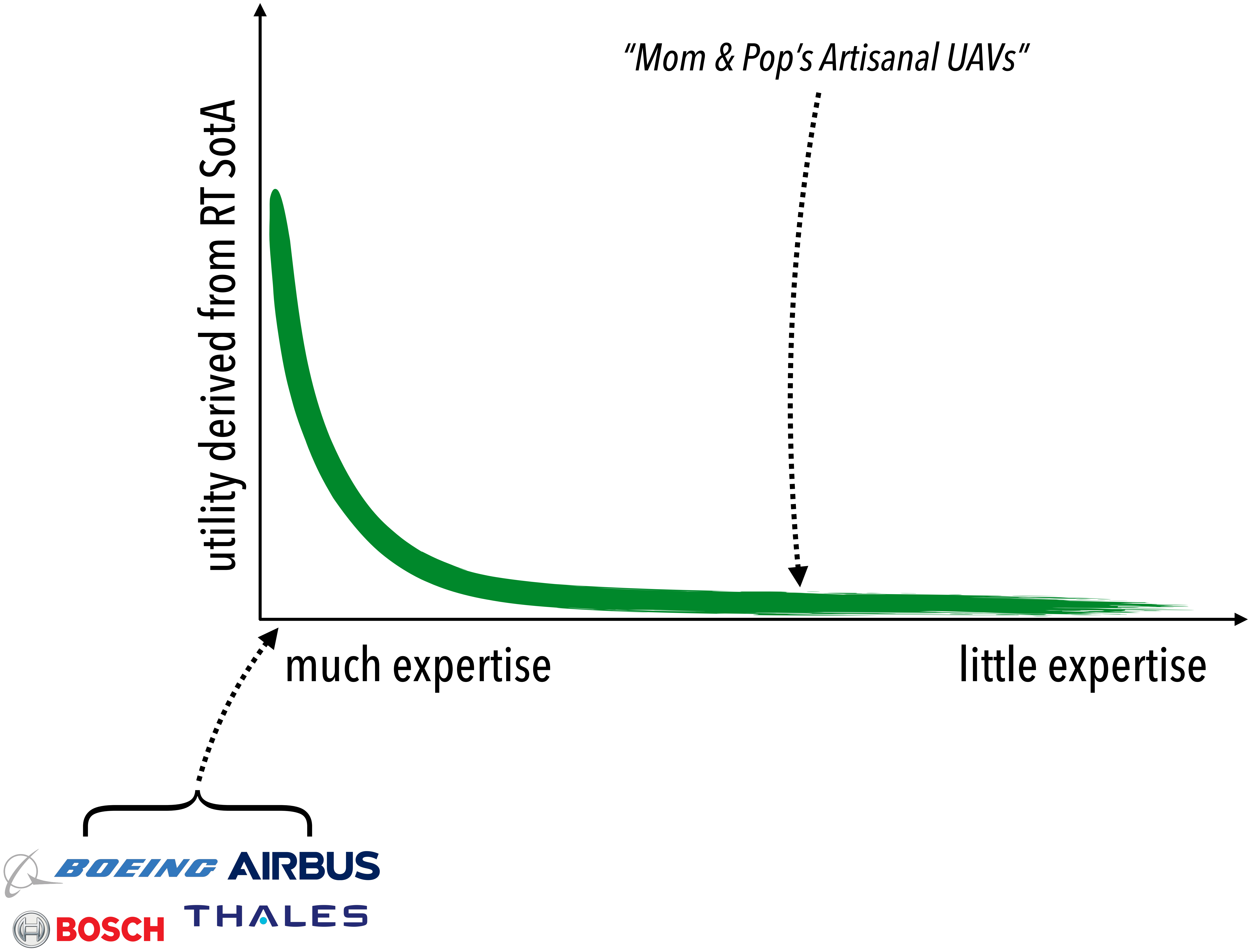
BUT WHAT ABOUT USERS IN THE "LONG TAIL"?



BUT WHAT ABOUT USERS IN THE "LONG TAIL"?

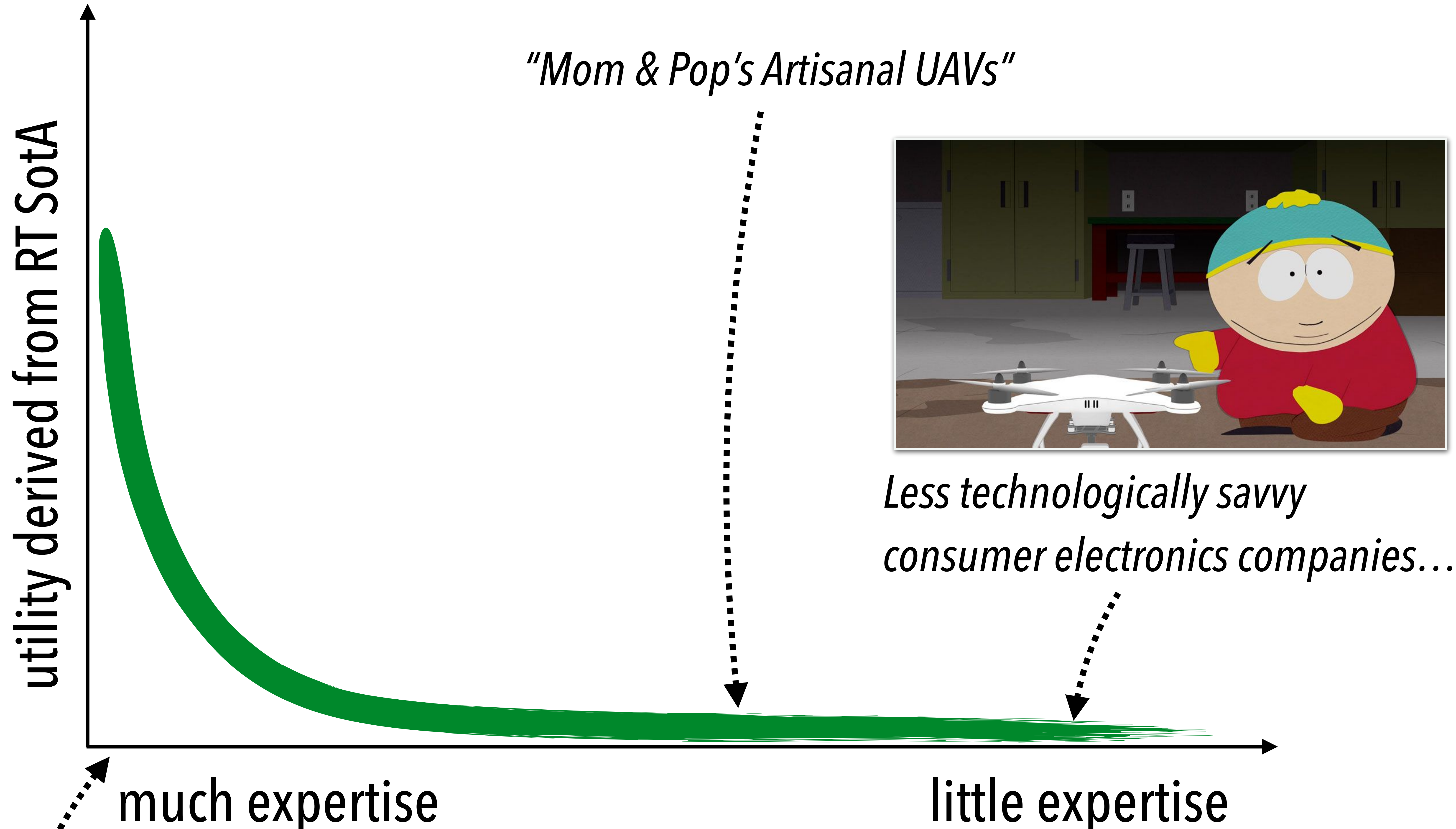


BUT WHAT ABOUT USERS IN THE "LONG TAIL"?



BUT WHAT ABOUT USERS IN THE "LONG TAIL"?

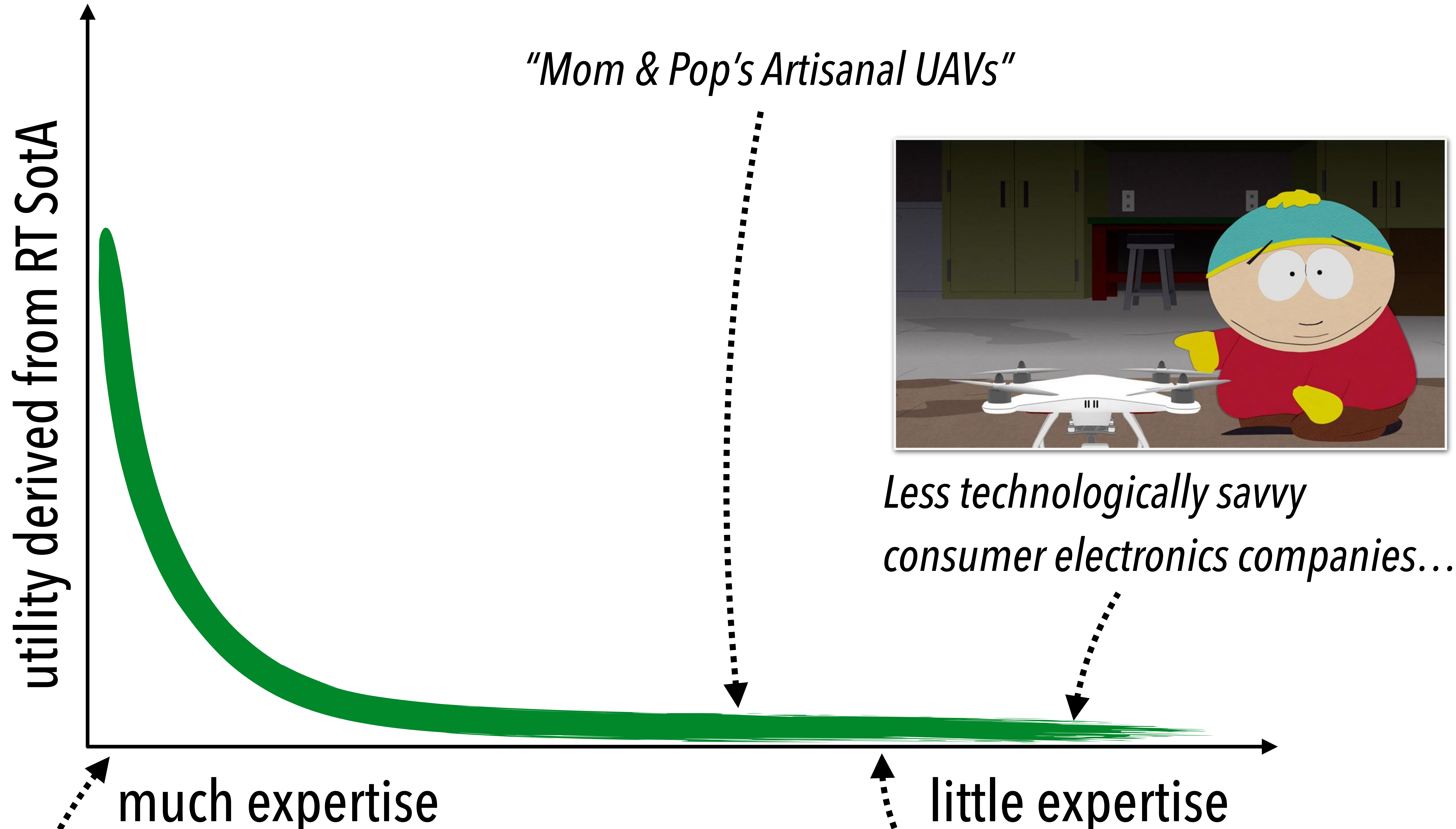
"Mom & Pop's Artisanal UAVs"



Less technologically savvy consumer electronics companies…

BOEING AIRBUS
BOSCH THALES

BUT WHAT ABOUT USERS IN THE "LONG TAIL"?



"Mom & Pop's Artisanal UAVs"

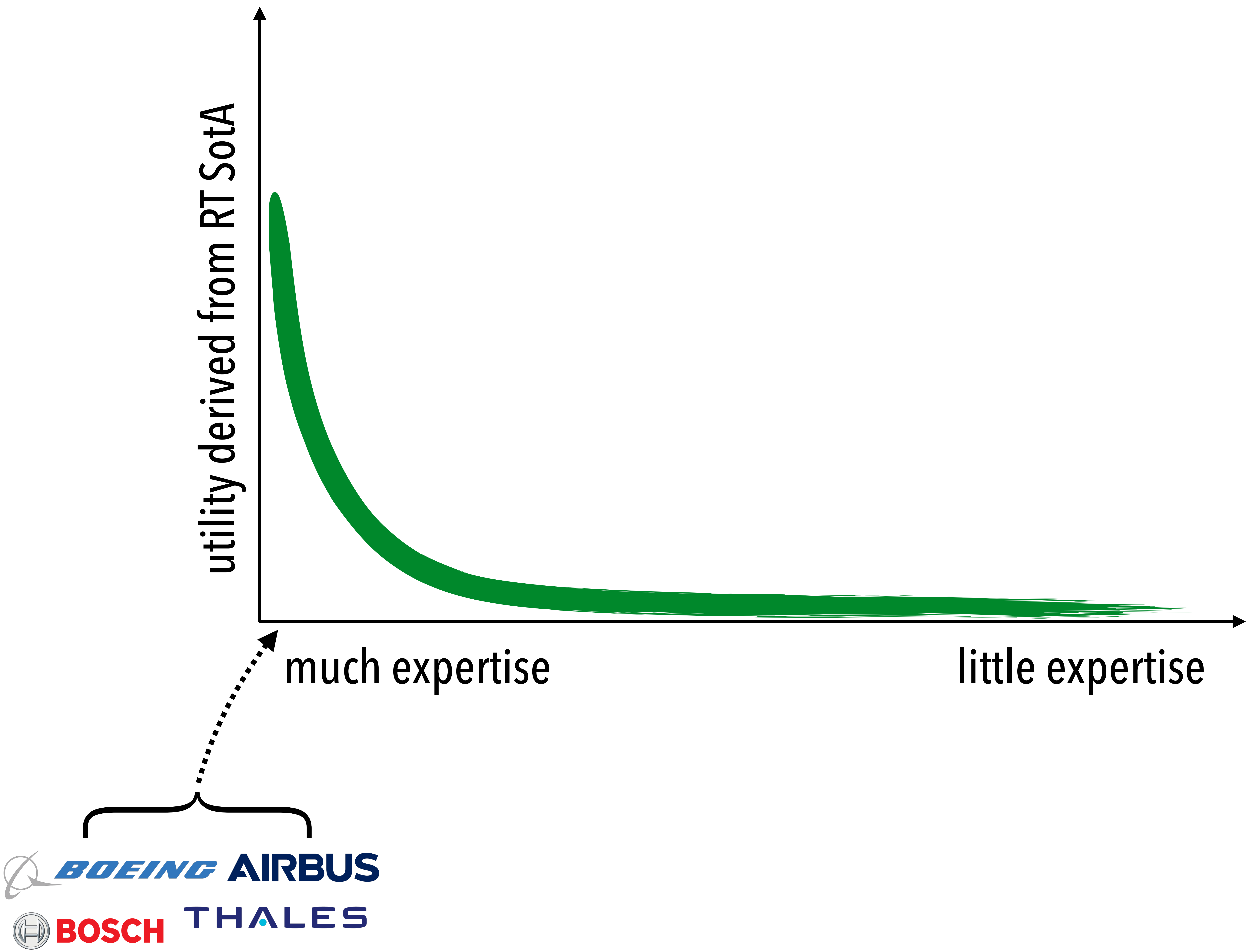
Less technologically savvy consumer electronics companies…

BOEING AIRBUS BOSCH THALES

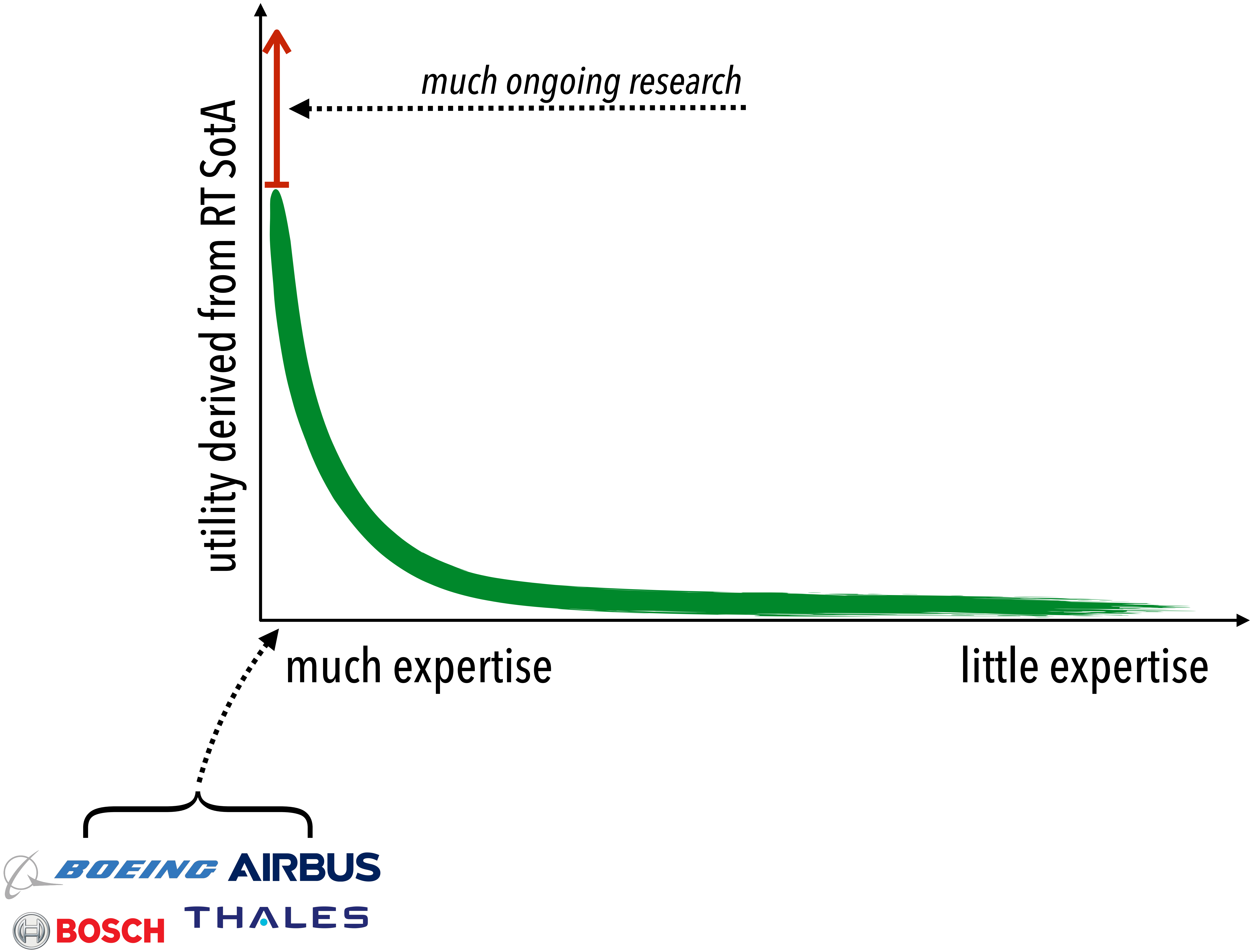
Download Linux & PREEMPT-RT & ROS

➔ *many applied robotics **researchers**…*

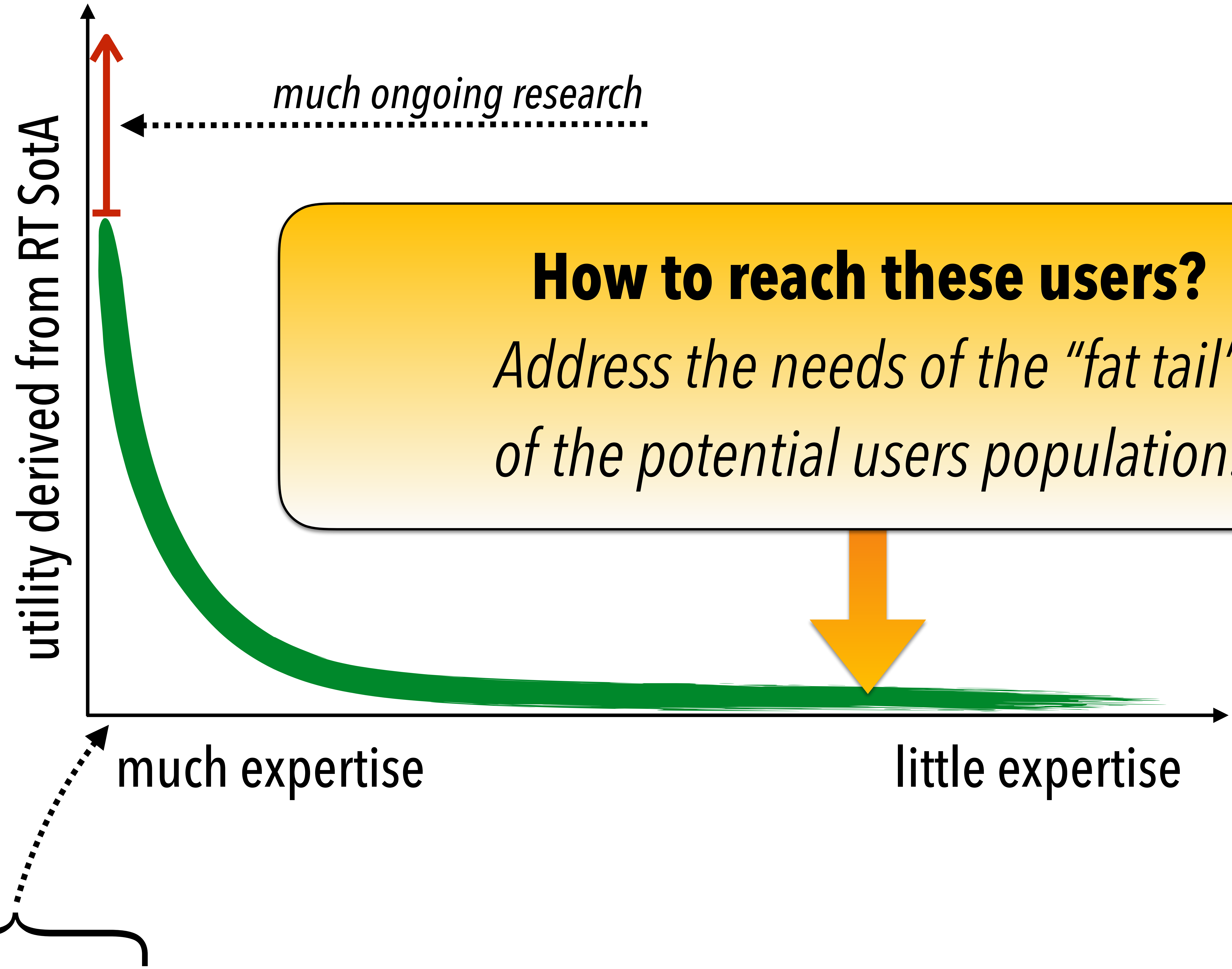
MOTIVATING OBSERVATION



MOTIVATING OBSERVATION



MOTIVATING OBSERVATION

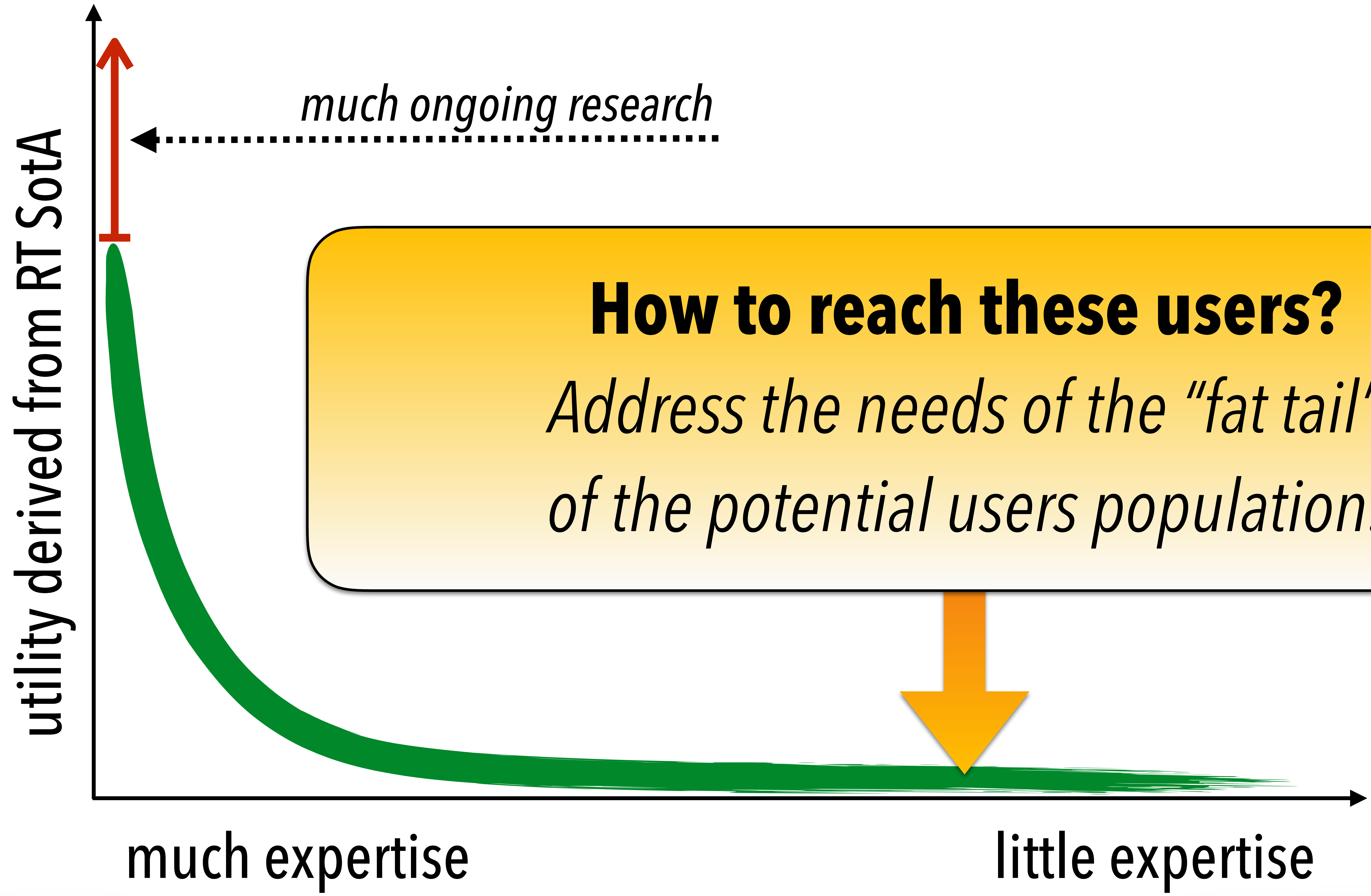


How to reach these users?

Address the needs of the "fat tail" of the potential users population.

BOEING AIRBUS
BOSCH THALES

MOTIVATING OBSERVATION



How to reach these users?

Address the needs of the "fat tail" of the potential users population.

Central Question

What prevents the widespread use of temporally sound system design?

HURDLES TO ADOPTION

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

priorities
affinities
semaphores
signals
pipes
sockets
…

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

priorities
affinities
semaphores
signals
pipes
sockets
…

combined in just the right way

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

priorities
affinities
semaphores
signals
pipes
sockets
…

combined in just the right way

In theory:
temporally sound, predictable system amenable to formal analysis

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

priorities
affinities
semaphores
signals
pipes
sockets
…

combined in just the right way

<u>In theory:</u>
temporally sound, predictable system amenable to formal analysis

If you know which pitfalls to avoid…

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

priorities
affinities
semaphores
signals
pipes
sockets
…

combined in just the right way

<u>In theory:</u>
temporally sound, predictable system amenable to formal analysis

If you know which pitfalls to avoid…

<u>In practice:</u>
*Domain experts are rarely also scheduling and timing analysis experts – **and why should they be?***

EXPERTISE BARRIER

*Current RTOSs expose mainly **low-level mechanisms** that are too difficult to use **correctly**.*

priorities
affinities
semaphores
signals
pipes
sockets
...

combined in just the right way

In theory:
temporally sound, predictable system amenable to formal analysis

If you know which pitfalls to avoid…

In practice:
*Domain experts are rarely also scheduling and timing analysis experts – **and why should they be?***

Don't need to be a compact flash expert to store a file…!
Don't need to be a concurrency control expert to query a database…!

COMPLEX TOOLING

Who wants to add "yet another tool" as a build dependency?

COMPLEX TOOLING

Who wants to add "yet another tool" as a build dependency?

Static Timing Analysis Tooling Today

$$$

and/or

not exactly user-friendly

and/or

difficult to integrate

and

static analysis is restrictive

COMPLEX TOOLING

Who wants to add "yet another tool" as a build dependency?

Static Timing Analysis Tooling Today

$$$

and/or

not exactly user-friendly

and/or

difficult to integrate

and

static analysis is restrictive

This may work for customers that **can't avoid** it…

…but it won't **entice users** in the "tail".

WHAT IF: `/proc/$PID/max-response-time—estimate`

WHAT IF: `/proc/$PID/max-response-time—estimate`

Suppose:

- ➔ wake-ups of `SCHED_FIFO` tasks **automatically** tracked
  - *over finite window (e.g., one second)*
- ➔ re-compute response-time bound whenever observations change
  - *based on **observed** peak arrivals and **observed** CPU consumption*
- ➔ does not require periodicity: can be represented as an *arrival curve*

WHAT IF: `/proc/$PID/max-response-time—estimate`

Suppose:

- ➔ wake-ups of `SCHED_FIFO` tasks **automatically** tracked
  - *over finite window (e.g., one second)*
- ➔ re-compute response-time bound whenever observations change
  - *based on **observed** peak arrivals and **observed** CPU consumption*
- ➔ does not require periodicity: can be represented as an *arrival curve*

Immediate use:

- ➔ online monitoring (top)
- ➔ adaptive system reconfiguration
- ➔ performance testing
- ➔ integration testing

WHAT IF: `/proc/$PID/max-response-time–estimate`

Suppose:

- ➔ wake-ups of `SCHED_FIFO` tasks **automatically** tracked
  - *over finite window (e.g., one second)*
- ➔ re-compute response-time bound whenever observations change
  - *based on* ***observed*** *peak arrivals and* ***observed*** *CPU consumption*
- ➔ does not require periodicity: can be represented as an *arrival curve*

Immediate use:

- ➔ online monitoring (top)
- ➔ adaptive system reconfiguration
- ➔ performance testing
- ➔ integration testing

Formal, sound timing analysis *based on estimated parameters "for free"!*

WHAT IF: `/proc/$PID/max-response-time—estimate`

Suppose:

➔ wake-ups of `SCHED_FIFO` tasks **automatically** tracked

- *over finite window (e.g., one second)*

➔ re-compute response-time bound whenever observations change

- *based on **observed** peak arrivals and **observed** CPU consumption*

➔ does not require periodicity: can be represented as an *arrival curve*

Immediate use:

➔ online monitoring (top)

➔ adaptive system reconfiguration

➔ performance testing

➔ integration testing

Formal, sound timing analysis *based on estimated parameters "for free"!*

➔ *much higher confidence from existing testing*

ADAPTIVE BELOW-WORST-CASE PROVISIONING

ADAPTIVE BELOW-WORST-CASE PROVISIONING

Standard Assumptions in the RT Literature

➔ static workload

➔ **worst-case execution times** (WCETs) known *a priori*

ADAPTIVE BELOW-WORST-CASE PROVISIONING

Standard Assumptions in the RT Literature

➔ static workload

➔ **worst-case execution times** (WCETs) known *a priori*

Reality

➔ many promising CPS applications are **inherently dynamic** (*e.g., robotics, autonomous vehicles, complex environments, …*)

➔ **cost-efficient** commodity multicore platforms ➔ ***no WCETs!***

➔ worst-case provisioning = **inefficient resource use** in the average case

ADAPTIVE BELOW-WORST-CASE PROVISIONING

Standard Assumptions in the RT Literature

➔ static workload

➔ **worst-case execution times** (WCETs) known *a priori*

Reality

➔ many promising CPS applications are **inherently dynamic** (*e.g., robotics, autonomous vehicles, complex environments, …*)

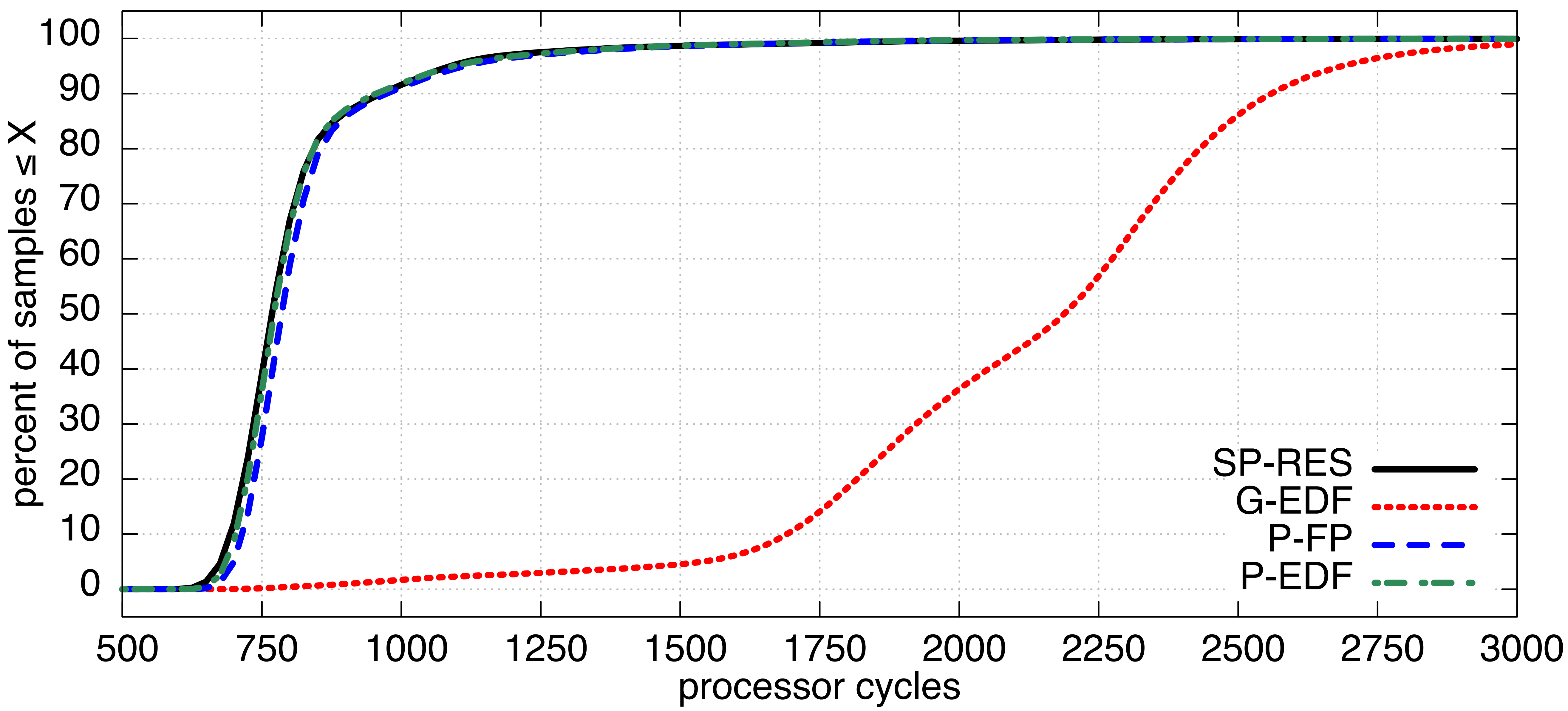
➔ **cost-efficient** commodity multicore platforms ➔ ***no WCETs!***

➔ worst-case provisioning = **inefficient resource use** in the average case

The essence of real-world engineering is graceful degradation

rather than static worst-case guarantees that are established once and then hold "forever".

EXAMPLE: CONTEXT-SWITCH OVERHEAD



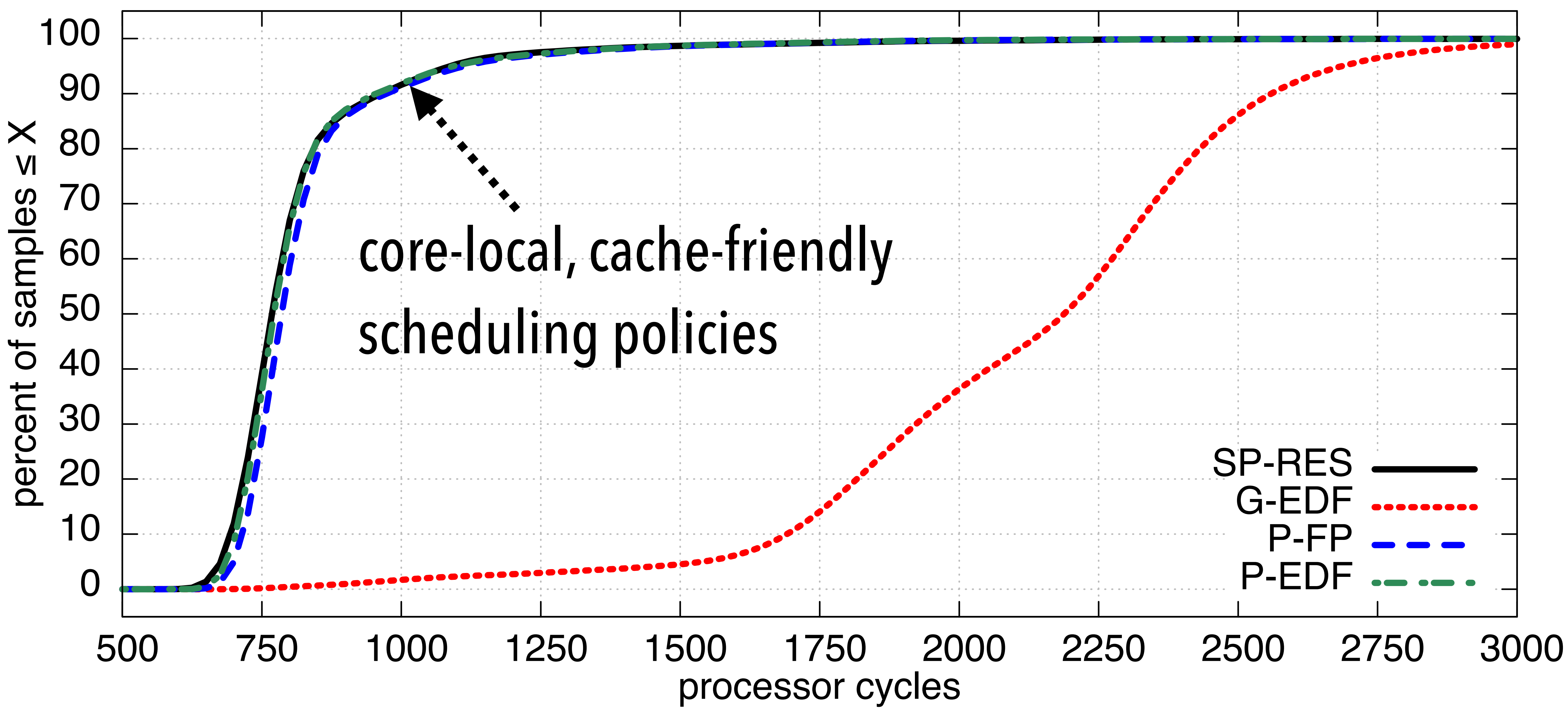
LITMUS^{RT}

Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Xeon E5-2699 v4 @ 2.2 GHz

[RTSS'16]

EXAMPLE: CONTEXT-SWITCH OVERHEAD

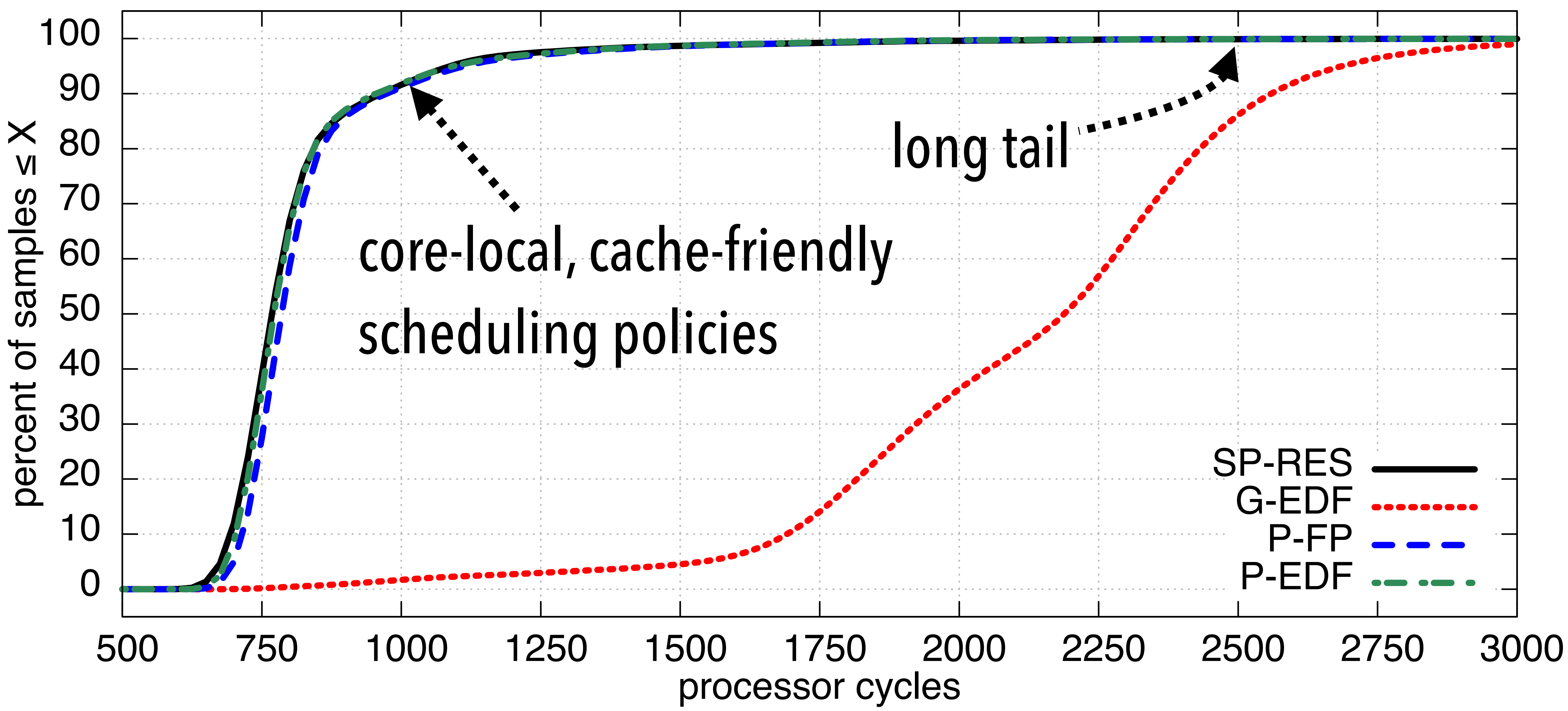


LITMUS RT
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Xeon E5-2699 v4 @ 2.2 GHz

[RTSS'16]

EXAMPLE: CONTEXT-SWITCH OVERHEAD

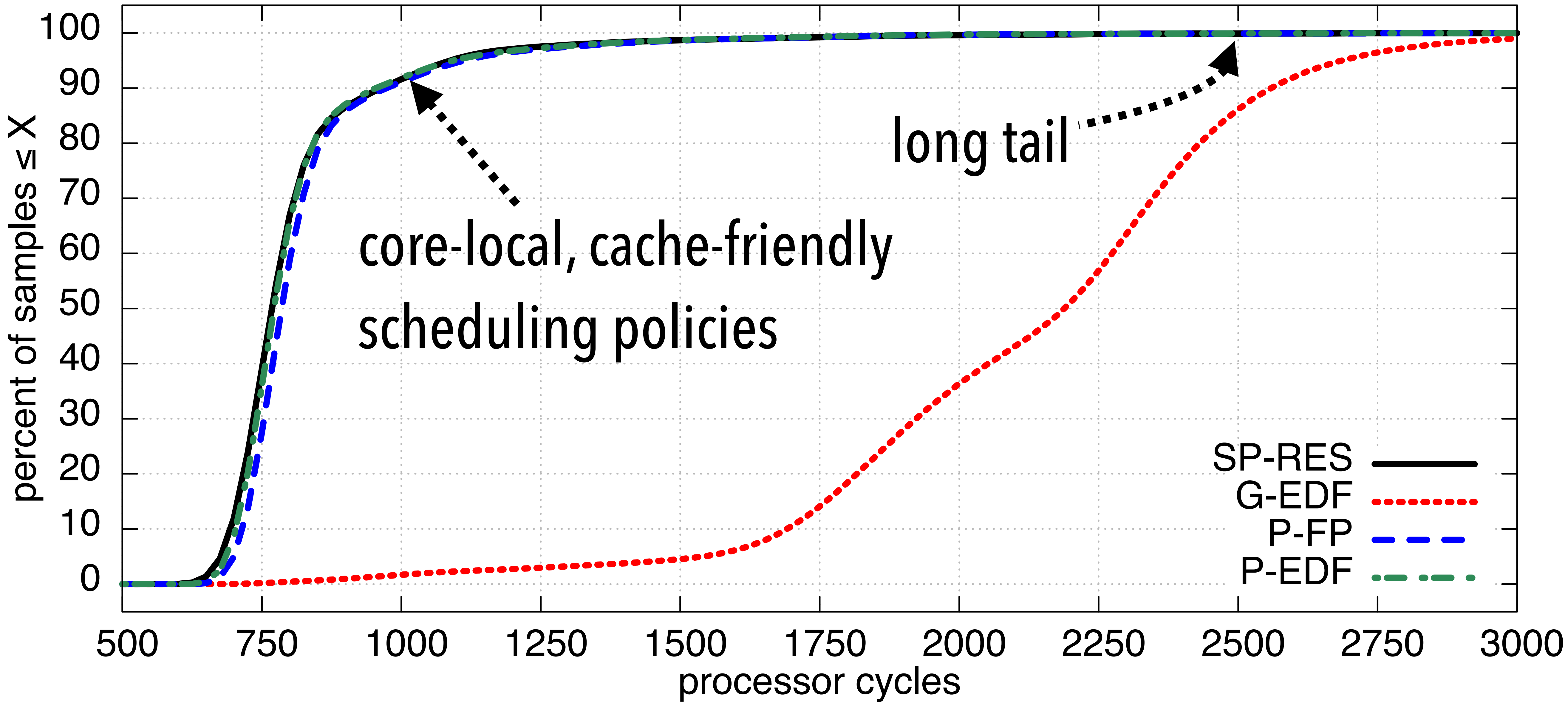


LITMUS^RT
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Xeon E5-2699 v4 @ 2.2 GHz

[RTSS'16]

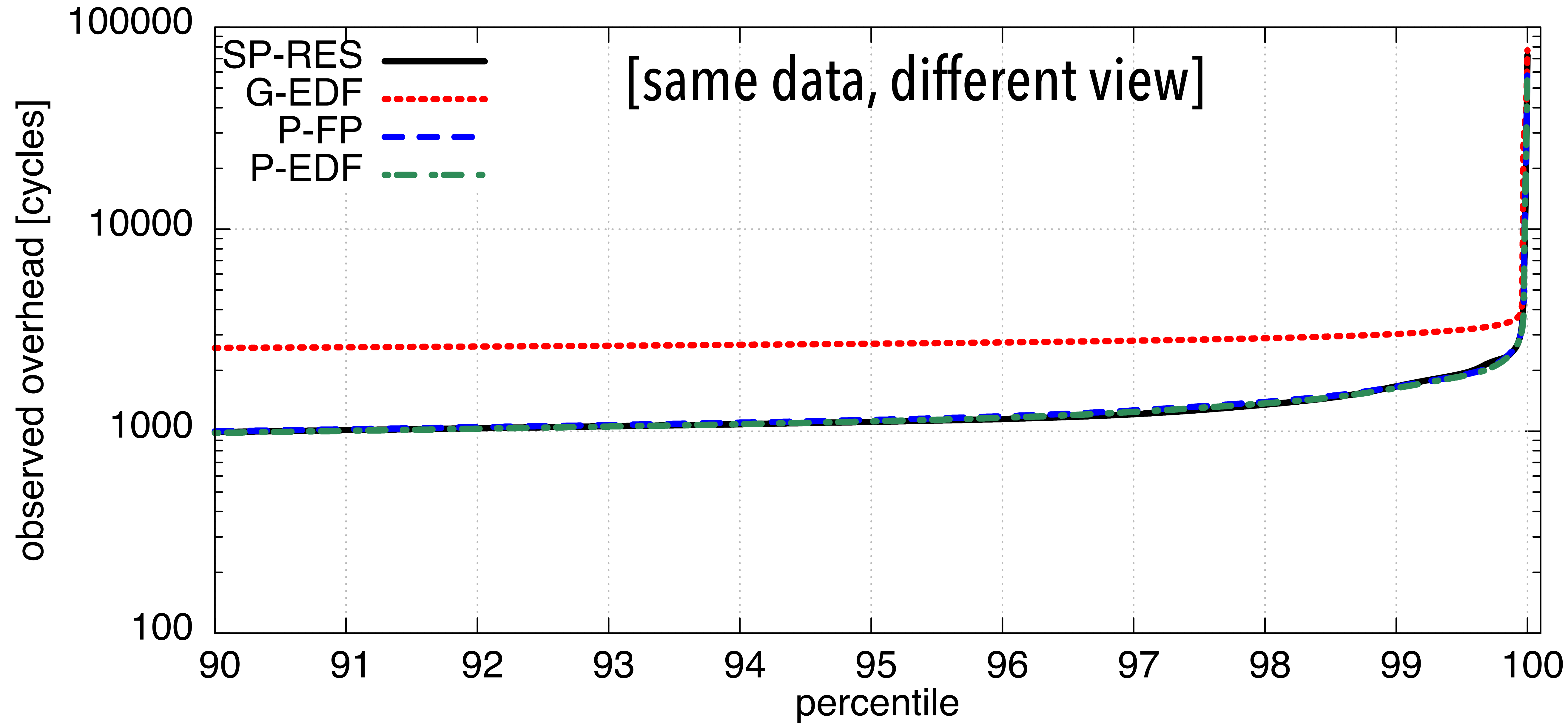
EXAMPLE: CONTEXT-SWITCH OVERHEAD



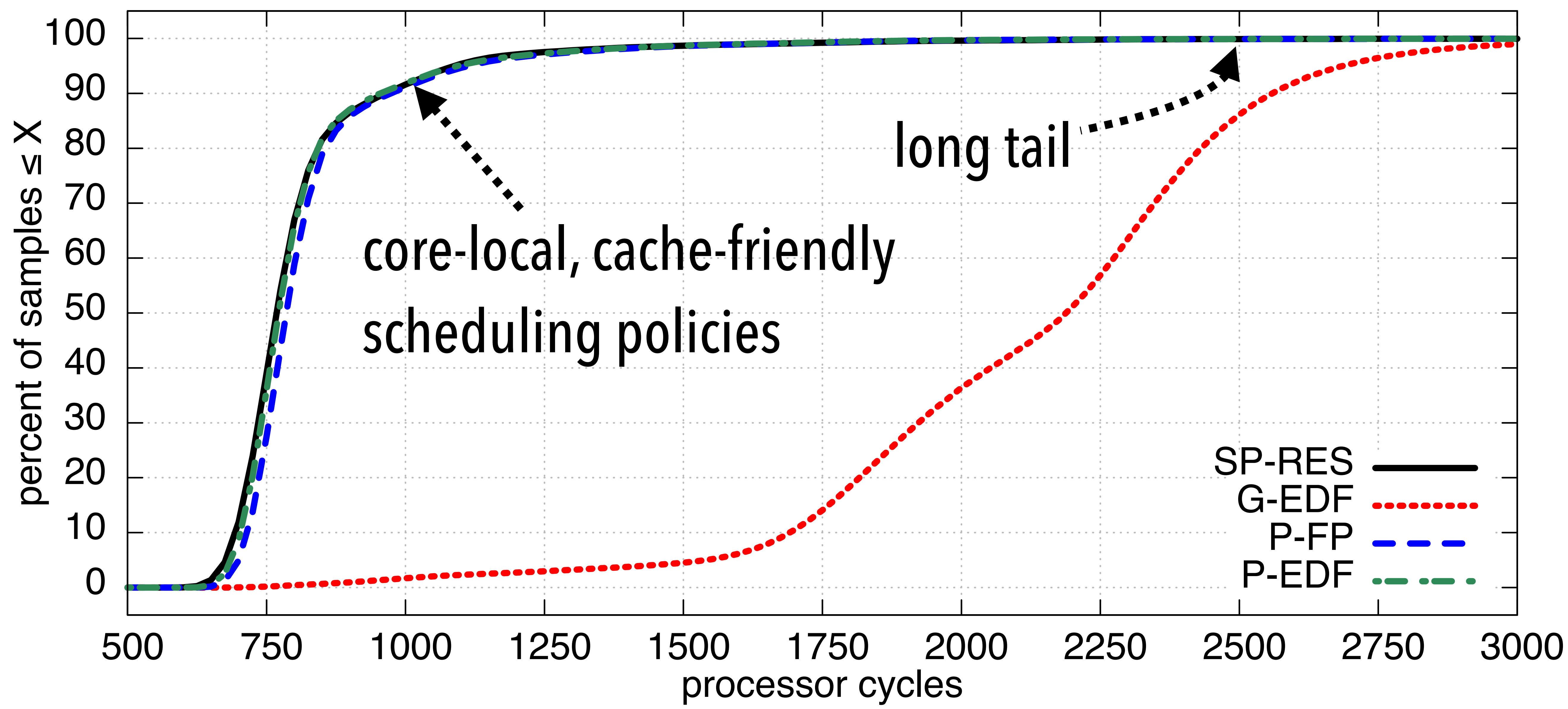
LITMUS^RT
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

Xeon E5-2699 v4 @ 2.2 GHz

[RTSS'16]



EXAMPLE: CONTEXT-SWITCH OVERHEAD

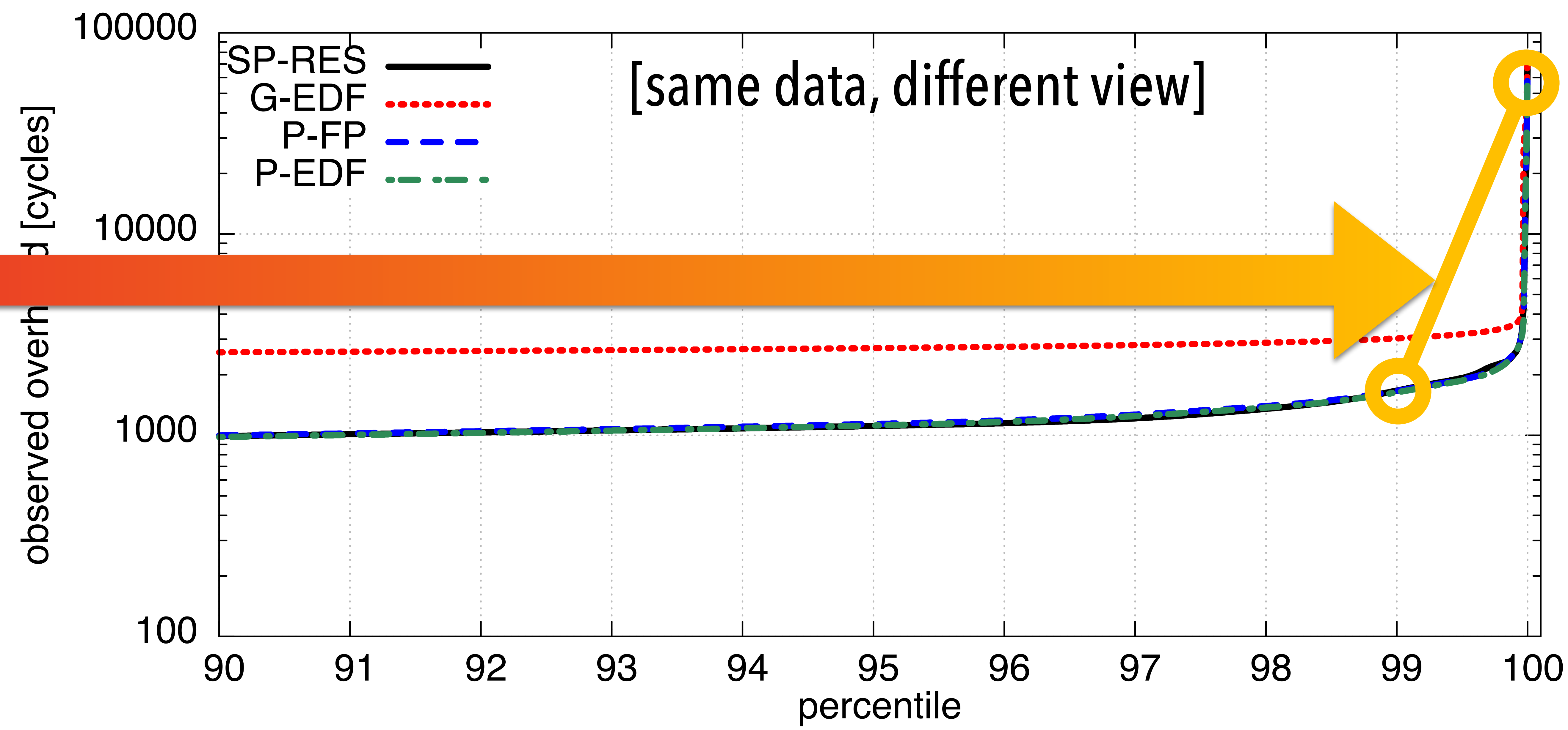


LITMUS^RT
Linux Testbed for Multiprocessor Scheduling in Real-Time Systems

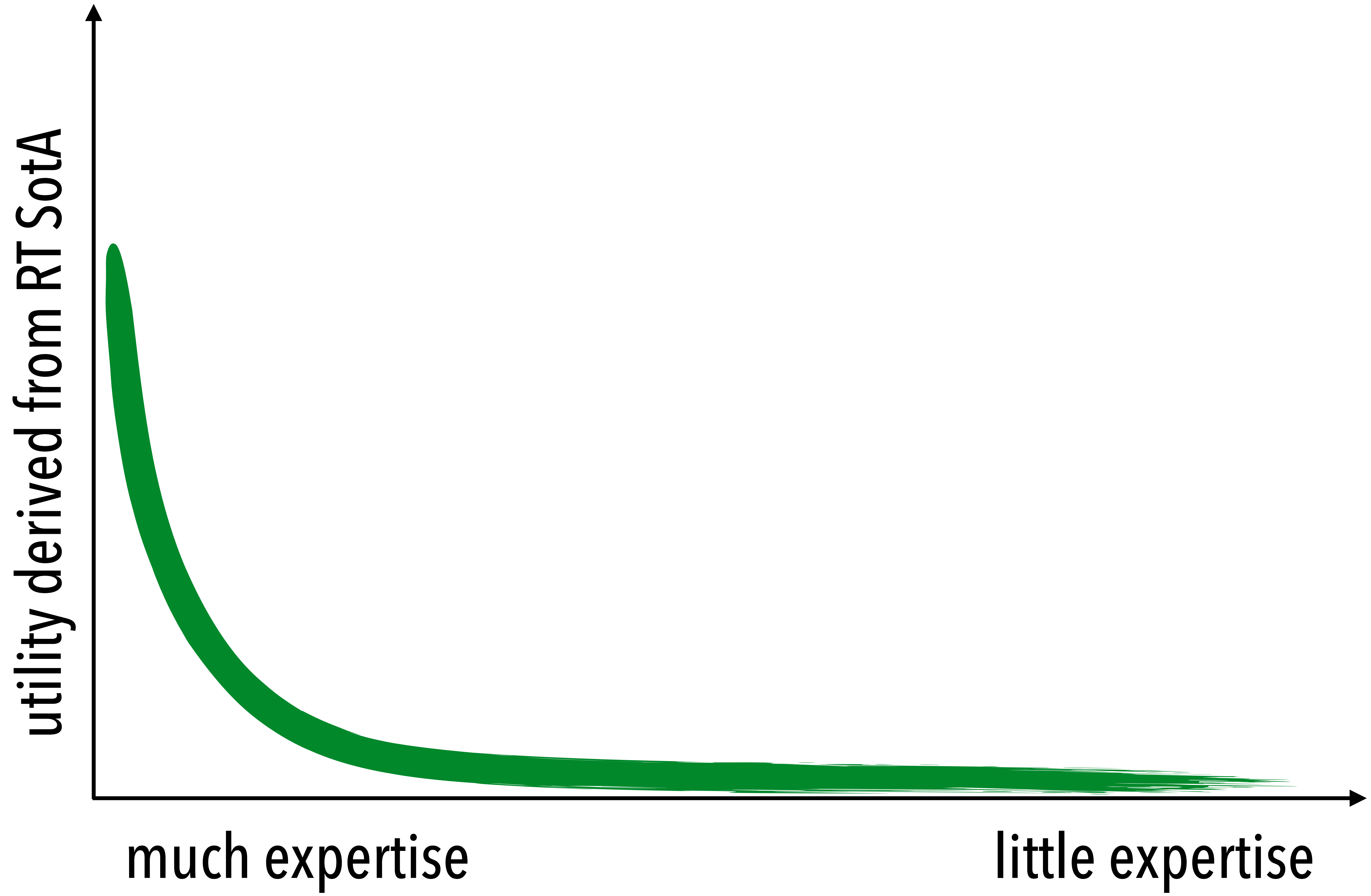
Xeon E5-2699 v4 @ 2.2 GHz

[RTSS'16]

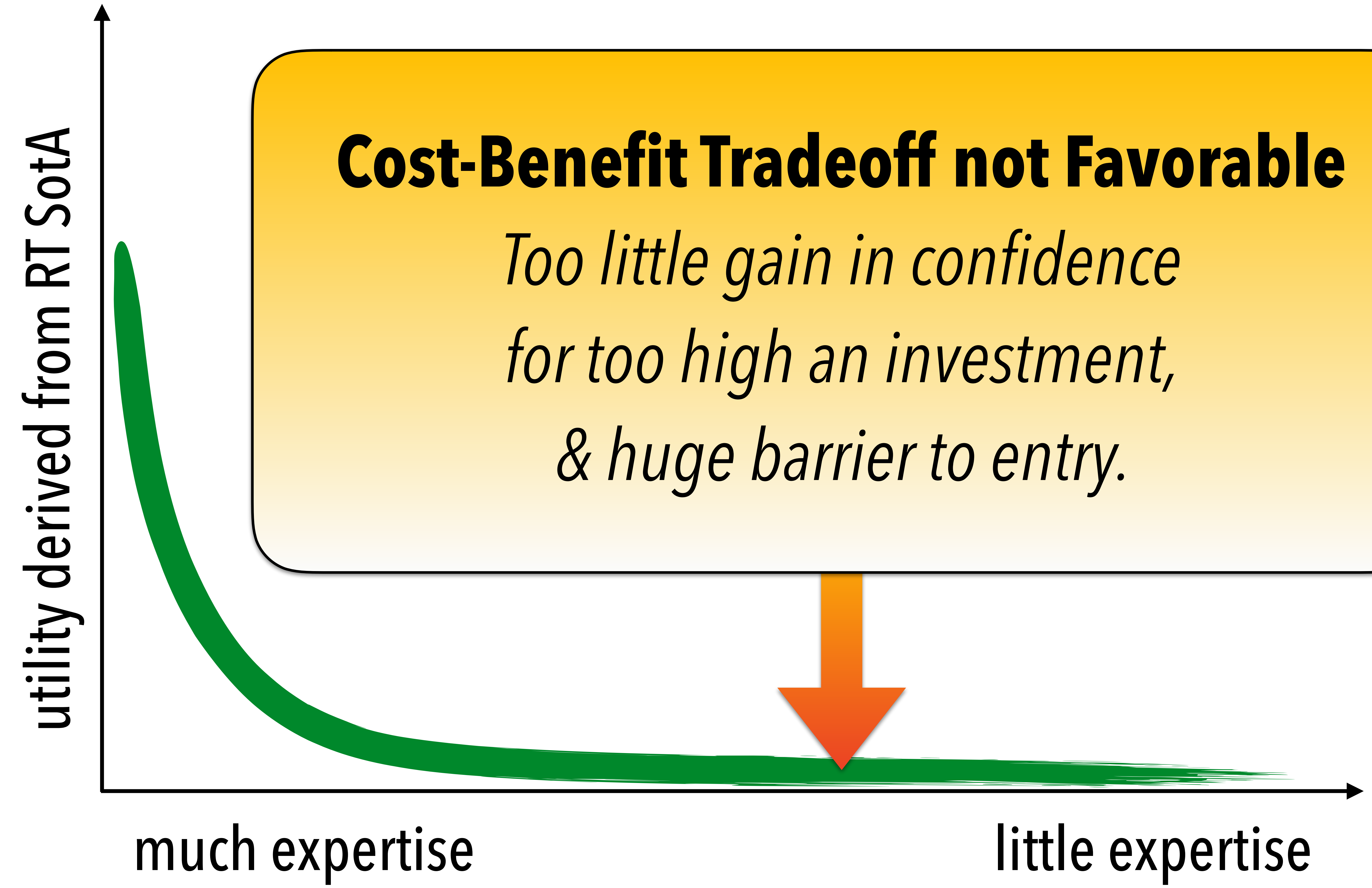
>10× delta
between 99th percentile
and ***observed*** *maximum!*



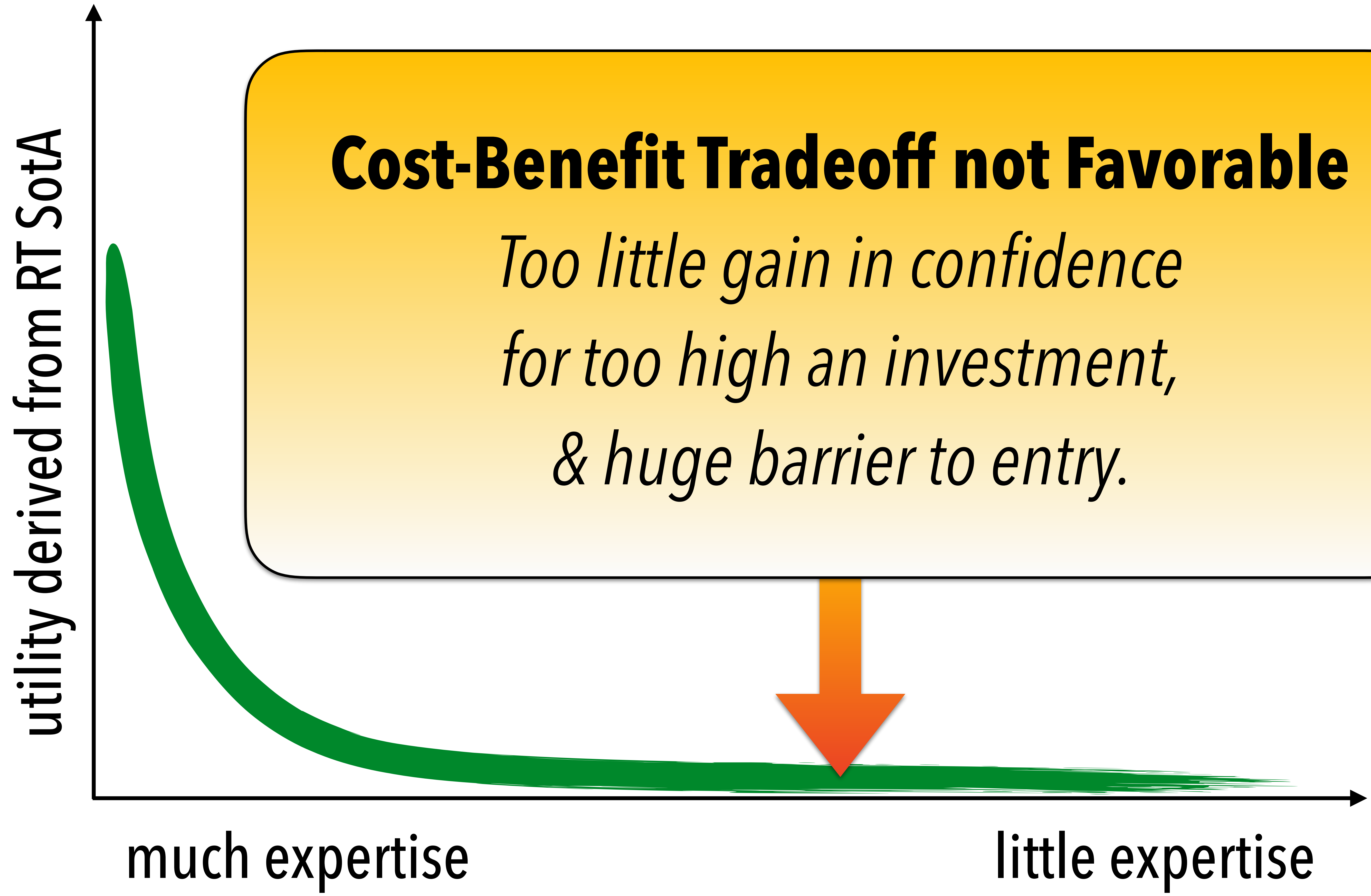
THE PROBLEM


utility derived from RT SotA
much expertise
little expertise

THE PROBLEM


utility derived from RT SotA
Cost-Benefit Tradeoff not Favorable
Too little gain in confidence
for too high an investment,
& huge barrier to entry.
much expertise
little expertise

THE PROBLEM

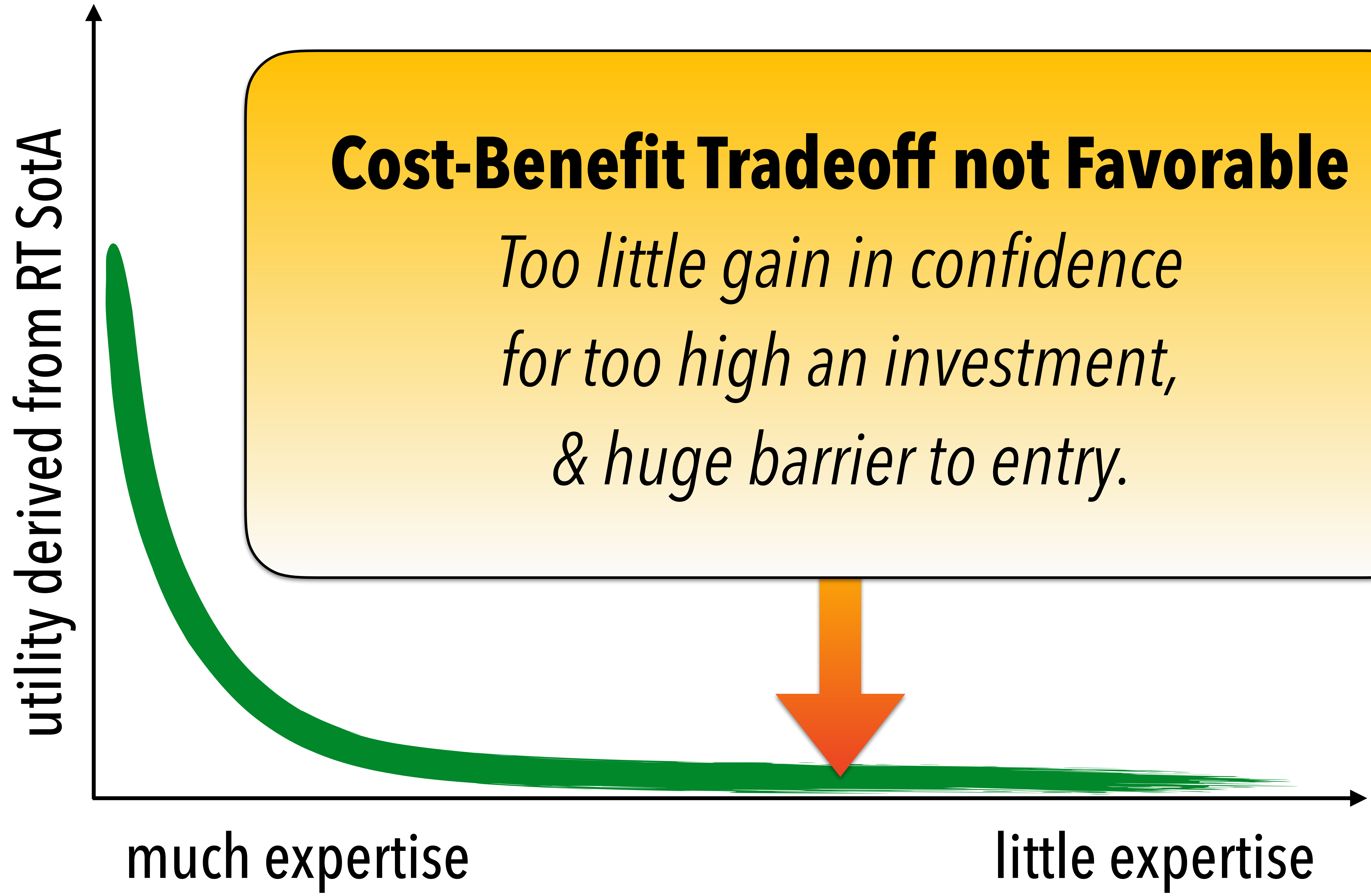


Cost-Benefit Tradeoff not Favorable

Too little gain in confidence
for too high an investment,
& huge barrier to entry.

Current **RTOSs** expose primarily difficult-to-use, **low-level mechanisms**

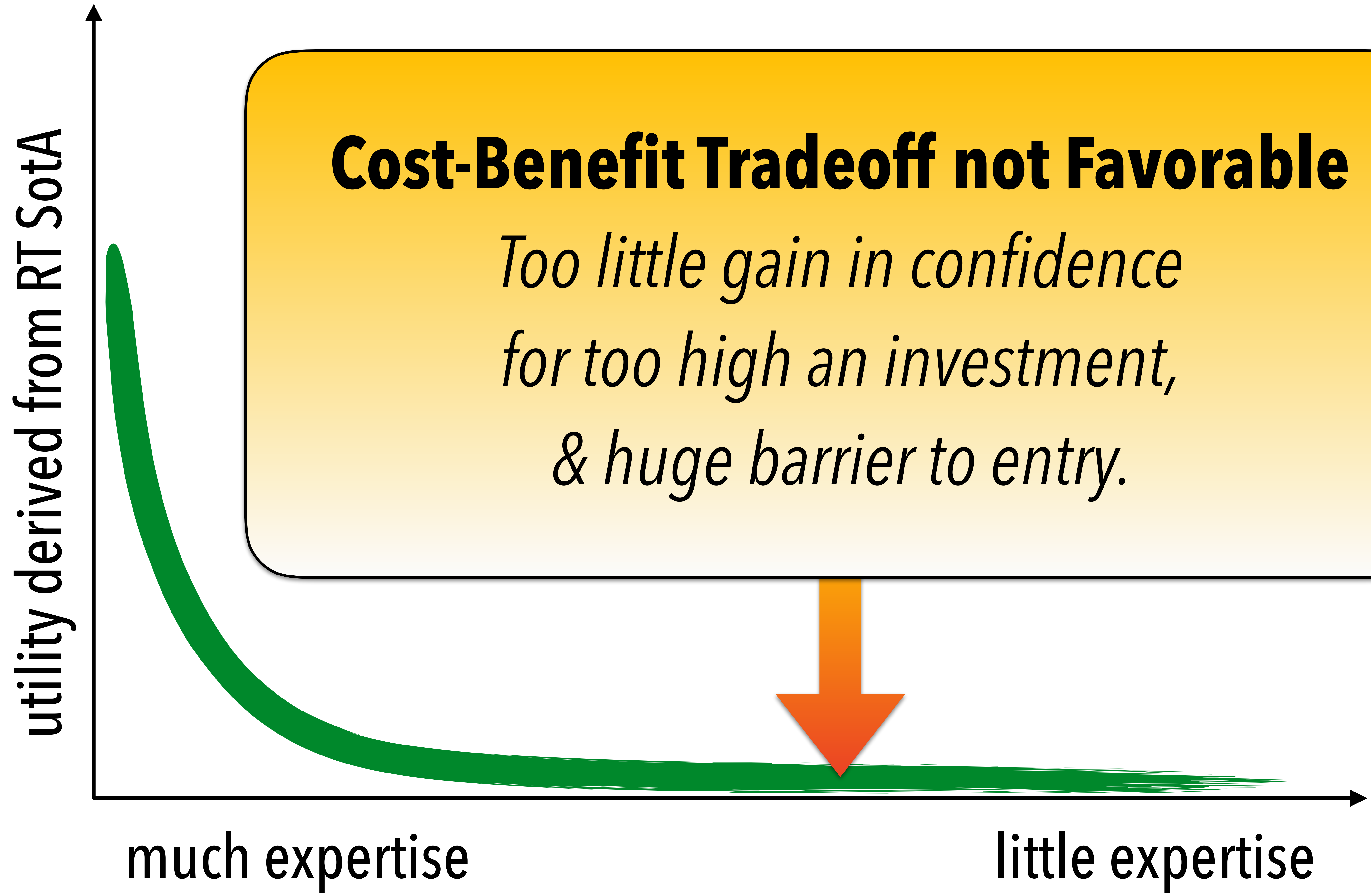
THE PROBLEM



Current **RTOSs** expose primarily difficult-to-use, **low-level mechanisms**

Current **analyses** rely too much on **highly idealized** worst-case assumptions

THE PROBLEM



Cost-Benefit Tradeoff not Favorable

Too little gain in confidence for too high an investment, & huge barrier to entry.

Current **RTOSs** expose primarily difficult-to-use, **low-level mechanisms**

Current **analyses** rely too much on **highly idealized** worst-case assumptions

Lack of confidence in soundness of complex analyses

Theory-
Oriented
Real-time
Operating
System

A
radically different,
practical foundation
for temporally sound
cyber-physical systems

European Research Council
Established by the European Commission

Theory-
Oriented
Real-time
Operating
System

A
radically different,
practical foundation
for temporally sound
cyber-physical systems

provably *free of timing errors*
(with high confidence)

European Research Council
Established by the European Commission

Theory-
Oriented
Real-time
Operating
System

*with **affordable** effort, under **realistic** assumptions*

A
radically different,
practical foundation
for temporally sound
cyber-physical systems

__provably__ free of timing errors (with high confidence)

Theory-
Oriented
Real-time
Operating
System

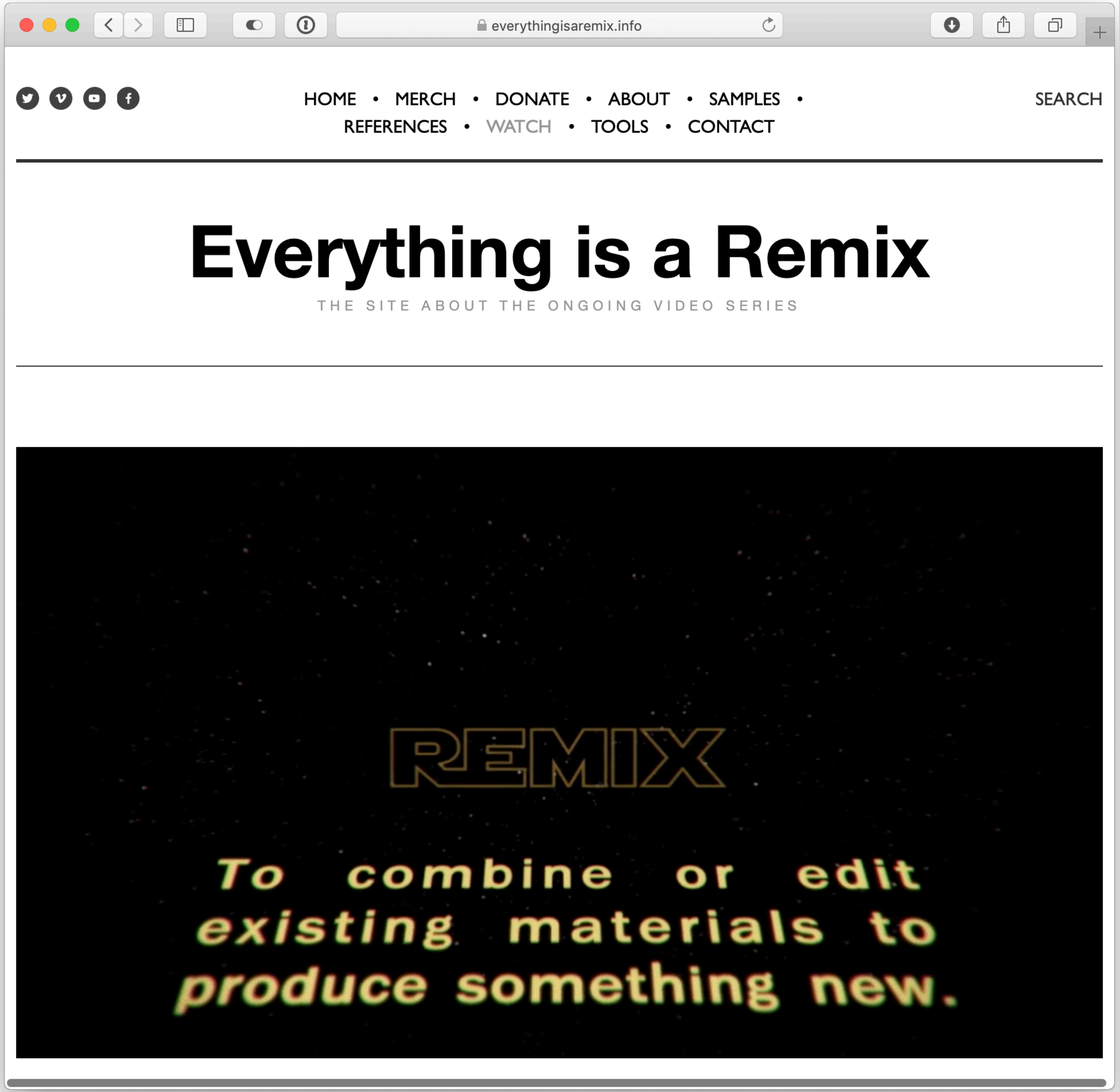
theory-first approach*: intersection of multiprocessor real-time scheduling theory and RTOS design*

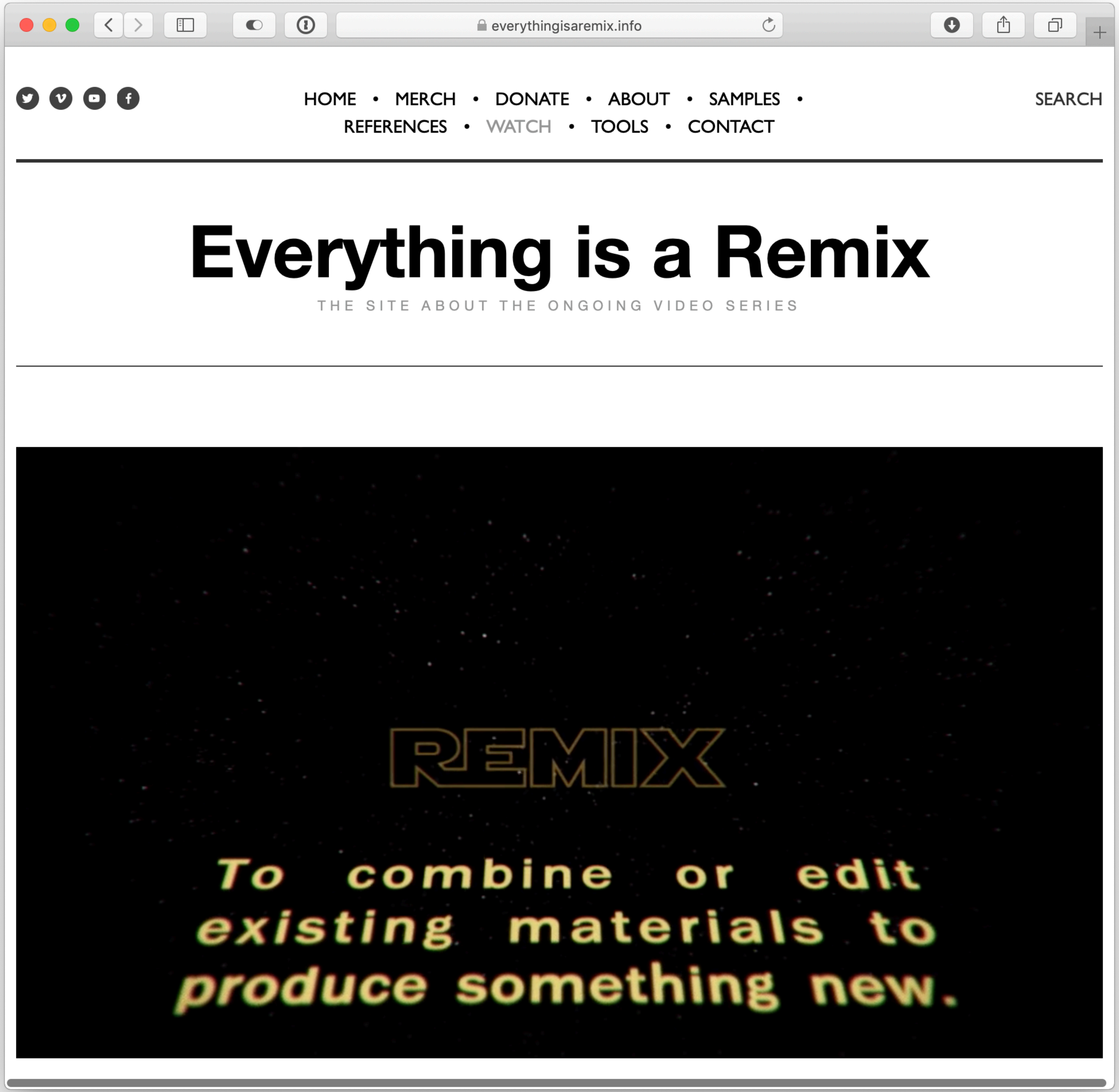
with ***affordable*** *effort, under* ***realistic*** *assumptions*

A
radically different,
practical foundation
for temporally sound
cyber-physical systems

provably *free of timing errors (with high confidence)*

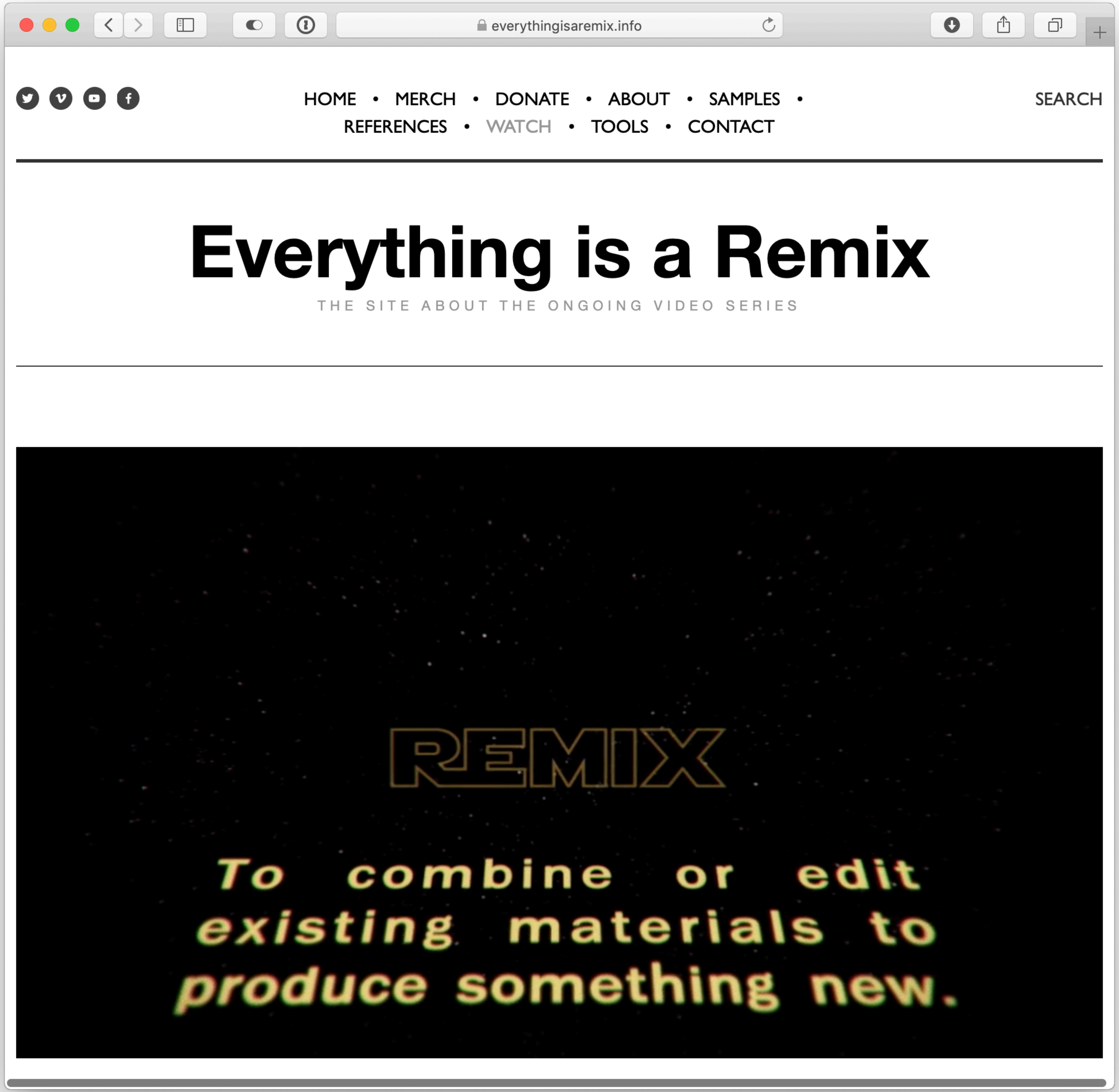
European Research Council
Established by the European Commission


everythingisaremix.info
HOME • MERCH • DONATE • ABOUT • SAMPLES •
REFERENCES • WATCH • TOOLS • CONTACT
SEARCH
Everything is a Remix
THE SITE ABOUT THE ONGOING VIDEO SERIES
REMIX
To combine or edit existing materials to produce something new.



Five Decades of RT Literature

➔ A *rich* foundation!

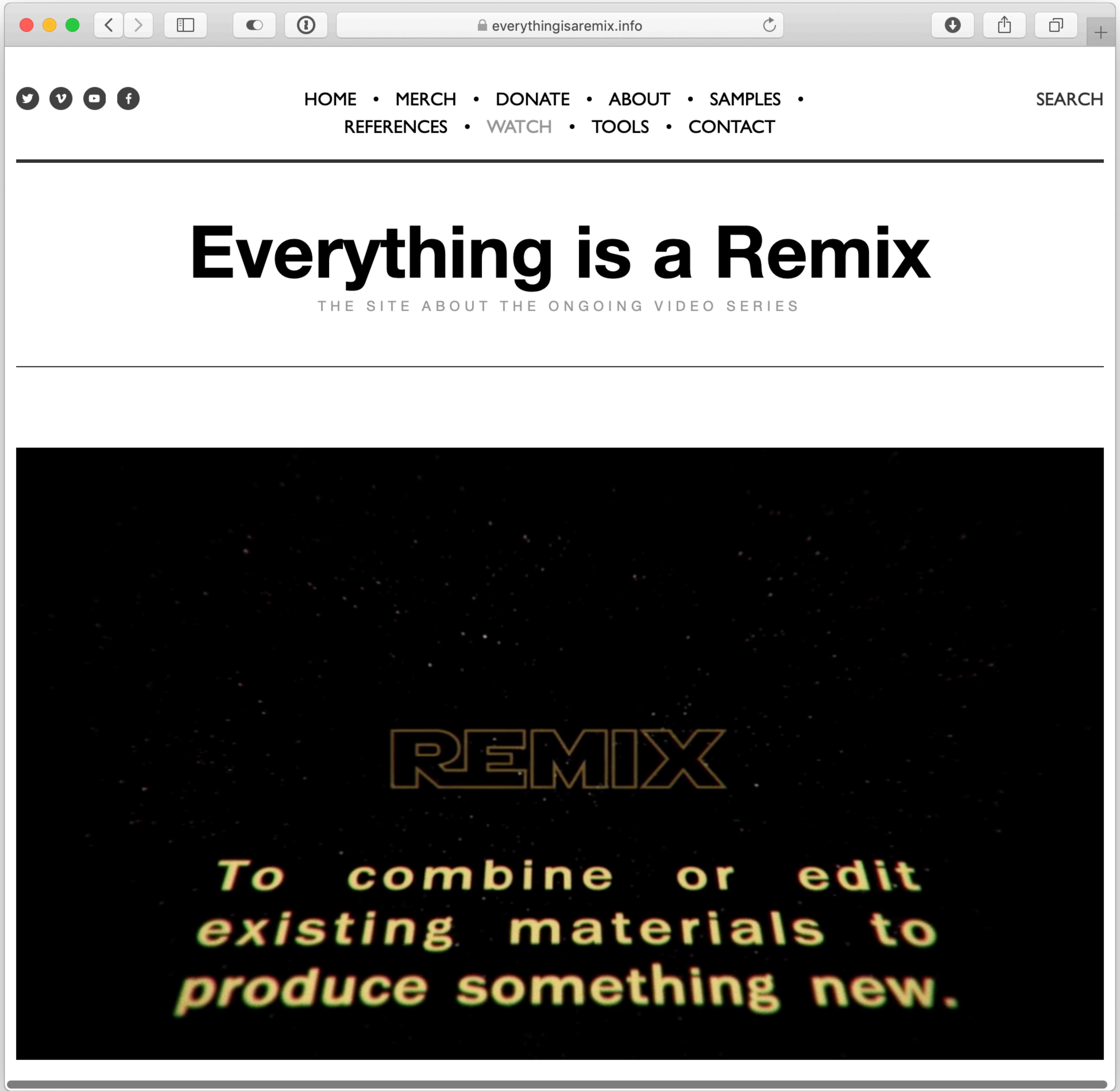


Five Decades of RT Literature

➔ A *rich* foundation!

No Need for Completely New Inventions

➔ Many great techniques that work to choose from



Five Decades of RT Literature

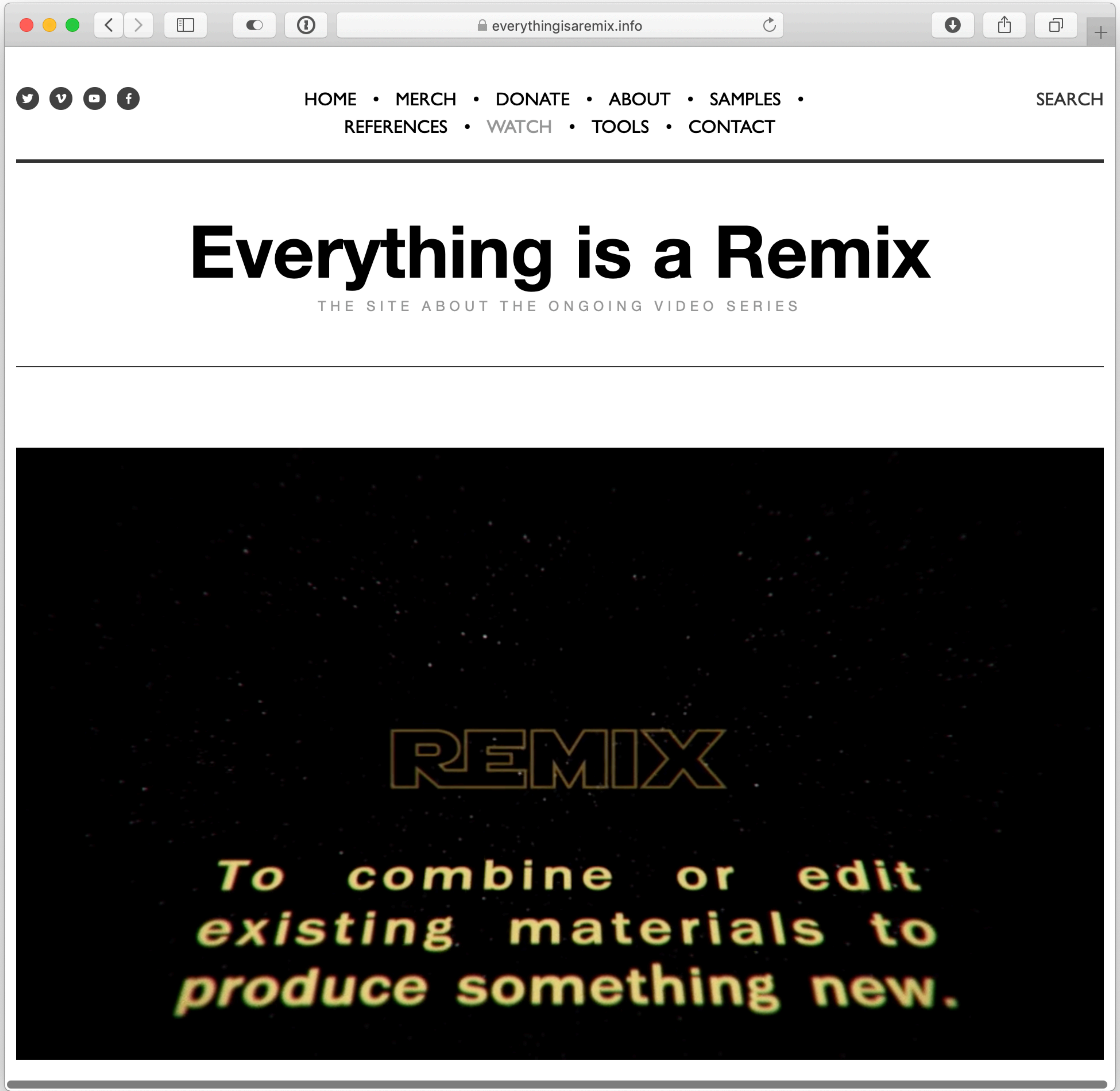
→ A *rich* foundation!

No Need for Completely New Inventions

→ Many great techniques that work to choose from

The Challenge

→ Select and combine just the right ideas in just the right way, *and remove the rest*



Five Decades of RT Literature

➔ A *rich* foundation!

No Need for Completely New Inventions

➔ Many great techniques that work to choose from

The Challenge

➔ Select and combine just the right ideas in just the right way, *and remove the rest*

The Promise

➔ More than the sum of its parts

THE FIVE TOROS PRINCIPLES

THE FIVE TOROS PRINCIPLES

(1) **theory-oriented RTOS design**: all provided abstractions must be **temporally sound** (➔ *any composition is analyzable*)

THE FIVE TOROS PRINCIPLES

(1) **theory-oriented RTOS design**: all provided abstractions must be **temporally sound** (➔ *any composition is analyzable*)

(2) **declarative OS abstractions**: automatically checkable timing and resource-allocation **goals** (➔ *domain experts do not need to understand scheduling*)

THE FIVE TOROS PRINCIPLES

(1) **theory-oriented RTOS design**: all provided abstractions must be **temporally sound** (➔ *any composition is analyzable*)

(2) **declarative OS abstractions**: automatically checkable timing and resource-allocation **goals** (➔ *domain experts do not need to understand scheduling*)

(3) **temporal reflection**: *transparently* & *continuously* self-assess temporal correctness and **proactively adapt** when guarantees can no longer be given

THE FIVE TOROS PRINCIPLES

(1) **theory-oriented RTOS design**: all provided abstractions must be **temporally sound** (➔ *any composition is analyzable*)

(2) **declarative OS abstractions**: automatically checkable timing and resource-allocation **goals** (➔ *domain experts do not need to understand scheduling*)

(3) **temporal reflection**: *transparently* & *continuously* self-assess temporal correctness and **proactively adapt** when guarantees can no longer be given

(4) **structured uncertainty management**: provide first-class, sound abstractions to manage uncertainty due to **below-worst-case provisioning**

THE FIVE TOROS PRINCIPLES

(1) **theory-oriented RTOS design**: all provided abstractions must be **temporally sound** (➔ *any composition is analyzable*)

(2) **declarative OS abstractions**: automatically checkable timing and resource-allocation **goals** (➔ *domain experts do not need to understand scheduling*)

(3) **temporal reflection**: *transparently* & *continuously* self-assess temporal correctness and **proactively adapt** when guarantees can no longer be given

(4) **structured uncertainty management**: provide first-class, sound abstractions to manage uncertainty due to **below-worst-case provisioning**

(5) **trustworthy analysis**: verify analysis soundness with **machine-checked proofs** using the **Coq** proof assistant [*not discussed today*]

(1) THEORY-ORIENTED OS DESIGN

all provided abstractions must be **temporally sound**
(➔ *any composition is analyzable*)

(1) THEORY-ORIENTED OS DESIGN

all provided abstractions must be **temporally sound**
(➔ *any composition is analyzable*)

<u>purity</u>
remove anything that isn't
(*including long-running processes!*)

(1) THEORY-ORIENTED OS DESIGN

all provided abstractions must be **temporally sound**
(➔ *any composition is analyzable*)

purity
remove anything that isn't
(*including long-running processes!*)

composition
allow only interactions with
backing scheduling theory

(1) THEORY-ORIENTED OS DESIGN

all provided abstractions must be **temporally sound**
(→ *any composition is analyzable*)

<u>purity</u>
remove anything that isn't
(*including long-running processes!*)

<u>composition</u>
allow only interactions with
backing scheduling theory

<u>no *accidental* unpredictability</u>
any idiomatic TOROS application
must always be analyzable

(1) THEORY-ORIENTED OS DESIGN

all provided abstractions must be **temporally sound**
(➔ *any composition is analyzable*)

purity
remove anything that isn't
(*including long-running processes!*)

composition
allow only interactions with
backing scheduling theory

no *accidental* unpredictability
any idiomatic TOROS application
must always be analyzable

Challenges

➔ Can one still build practical applications with reasonable effort on top of such a minimal, unconventional foundation?

➔ Massive engineering effort to get the system off the ground…

(2) DECLARATIVE, HIGH-LEVEL OS ABSTRACTIONS

automatically checkable timing and resource-allocation **goals**
(➔ *domain experts do not need to understand real-time theory*)

(2) DECLARATIVE, HIGH-LEVEL OS ABSTRACTIONS

automatically checkable timing and resource-allocation **goals**
(➔ *domain experts do not need to understand real-time theory*)

do not expose scheduling policy
no priorities, no processor affinity, no detailed process configuration…

(2) DECLARATIVE, HIGH-LEVEL OS ABSTRACTIONS

automatically checkable timing and resource-allocation **goals**

(➔ *domain experts do not need to understand real-time theory*)

<u>do not expose scheduling policy</u>

no priorities, no processor affinity, no detailed process configuration…

<u>component-based systems</u>

strong time and space isolation + declarative timing goals

(2) DECLARATIVE, HIGH-LEVEL OS ABSTRACTIONS

automatically checkable timing and resource-allocation **goals**
(➔ *domain experts do not need to understand real-time theory*)

<u>do not expose scheduling policy</u>
no priorities, no processor affinity, no detailed process configuration…

<u>component-based systems</u>
strong time and space isolation + declarative timing goals

<u>automatic synchronization</u>
no locking primitives ➔ occupancy constraints [monitors]

(2) DECLARATIVE, HIGH-LEVEL OS ABSTRACTIONS

automatically checkable timing and resource-allocation **goals**
(➔ *domain experts do not need to understand real-time theory*)

<u>do not expose scheduling policy</u>
no priorities, no processor affinity, no detailed process configuration…

<u>component-based systems</u>
strong time and space isolation + declarative timing goals

<u>automatic synchronization</u>
no locking primitives ➔ *occupancy constraints [monitors]*

Challenges

➔ Need one-size-fits-all scheduling and synchronization policies

➔ Automatically map specified goals to efficient parameter choices

BASIC ABSTRACTIONS IN TOROS

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

spatial isolation

Logic & Data Compartment (LDC)

passive address space +
text + heap + bss +
entry points ("gates")

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

spatial isolation

Logic & Data Compartment (LDC)

passive address space +
text + heap + bss +
entry points ("gates")

execution management

Ephemeral Jobs (EJs)

short-lived, nameless
execution context (stack) +
run-to-completion semantics
(cannot suspend to wait)

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

➔ (α, Δ) bounded delay model [Mok et al., RTAS'01]

➔ realizable with **near-optimal** semi-partitioned reservations [RTSS'16]

spatial isolation

Logic & Data Compartment (LDC)

passive address space + text + heap + bss + **entry points ("gates")**

execution management

Ephemeral Jobs (EJs)

short-lived, nameless execution context (stack) + **run-to-completion semantics** (cannot suspend to wait)

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

- ➔ (α, Δ) bounded delay model [Mok et al., RTAS'01]
- ➔ realizable with **near-optimal** semi-partitioned reservations [RTSS'16]

spatial isolation

Logic & Data Compartment (LDC)

passive address space +
text + heap + bss +
entry points ("gates")

- ➔ components in Composite OS [Parmer, 2010]
- ➔ passive = not necessarily inherited by a thread
- ➔ entry points = like system calls
- ➔ like objects in an OO language

execution management

Ephemeral Jobs (EJs)

short-lived, nameless
execution context (stack) +
run-to-completion semantics
(cannot suspend to wait)

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

- ➔ (α, Δ) bounded delay model [Mok et al., RTAS'01]
- ➔ realizable with **near-optimal** semi-partitioned reservations [RTSS'16]

spatial isolation

Logic & Data Compartment (LDC)

passive address space + text + heap + bss + **entry points ("gates")**

- ➔ components in Composite OS [Parmer, 2010]
- ➔ passive = not necessarily inherited by a thread
- ➔ entry points = like system calls
- ➔ like objects in an OO language

execution management

Ephemeral Jobs (EJs)

short-lived, nameless execution context (stack) + **run-to-completion semantics** (cannot suspend to wait)

- ➔ like callbacks or event handlers in flight
- ➔ always start at some entry point
- ➔ cannot wait
- ➔ cannot be referenced

BASIC ABSTRACTIONS IN TOROS

temporal isolation

Guaranteed Processor Share (GPS)

Fraction of a core (%)
+
max. scheduling latency

spatial isolation

Logic & Data Compartment (LDC)

passive address space +
text + heap + bss +
entry points ("gates")

execution management

Ephemeral Jobs (EJs)

short-lived, nameless
execution context (stack) +
run-to-completion semantics
(cannot suspend to wait)

Only two ways for EJs to interact

`asynchronous_invoke(LDC::entry_point, GPS)` → **fork**

`synchronous_invoke(LDC::entry_point, GPS)` → **call-return semantics**

EXECUTION & PROGRAMMING MODEL

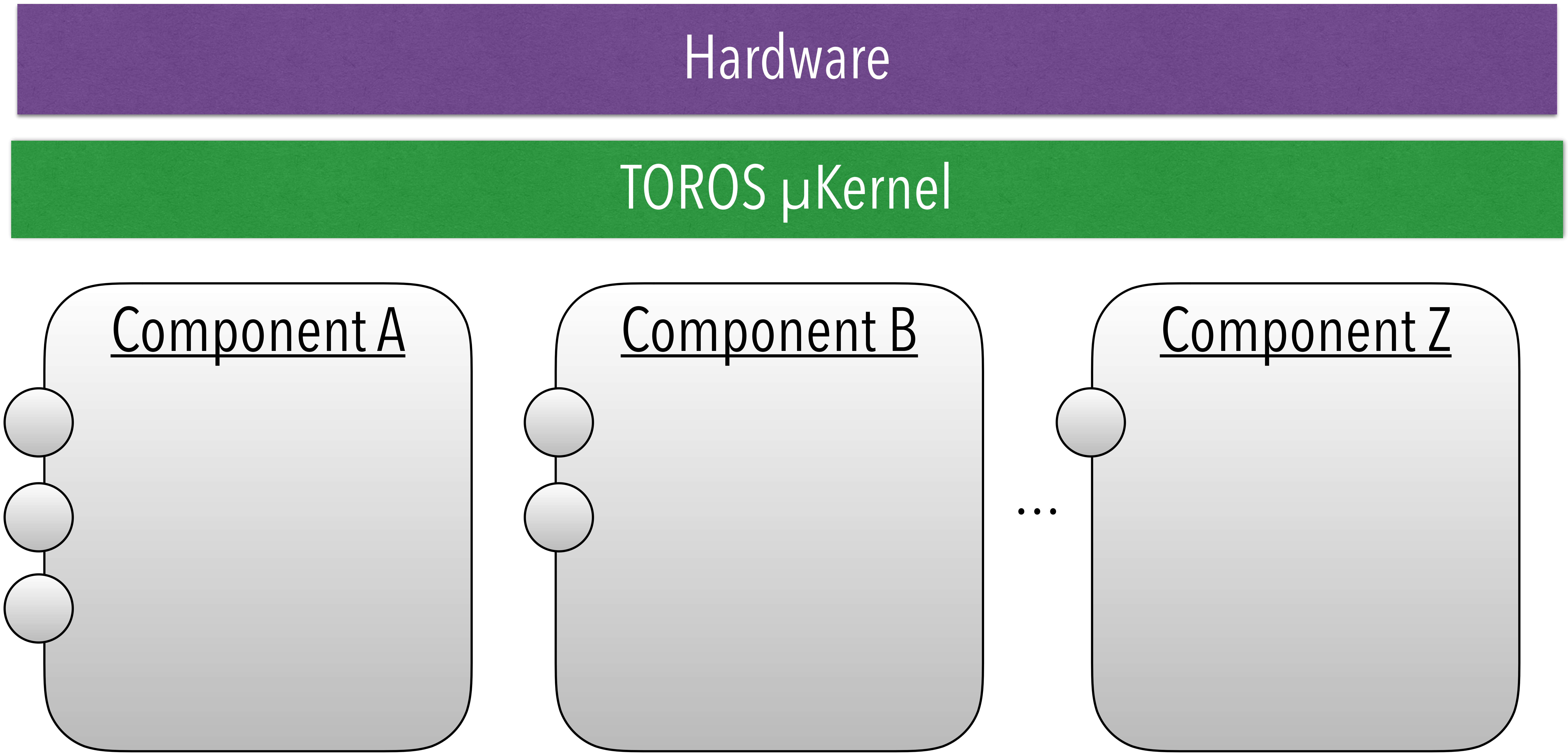
Hardware

EXECUTION & PROGRAMMING MODEL

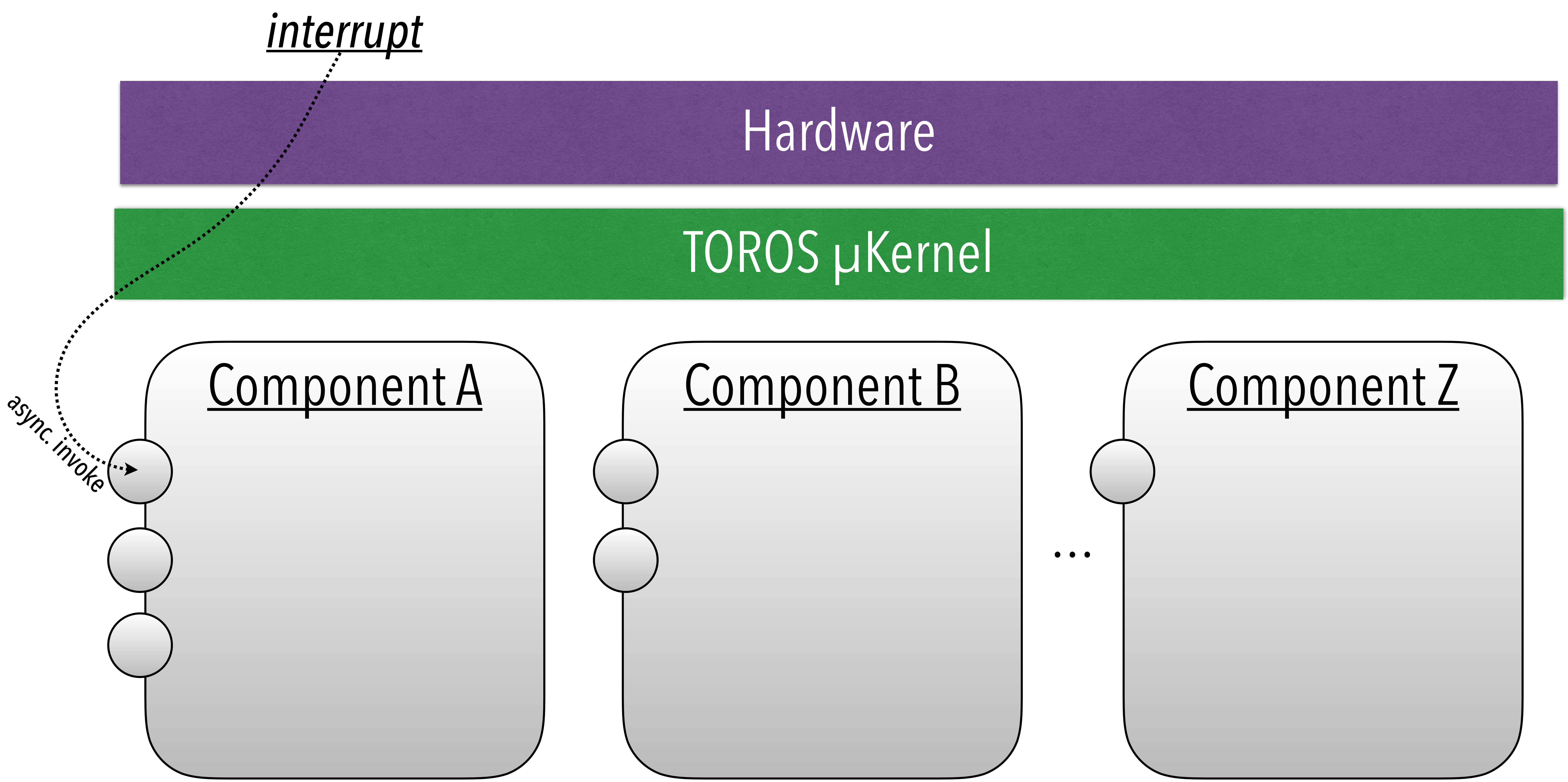
Hardware

TOROS µKernel

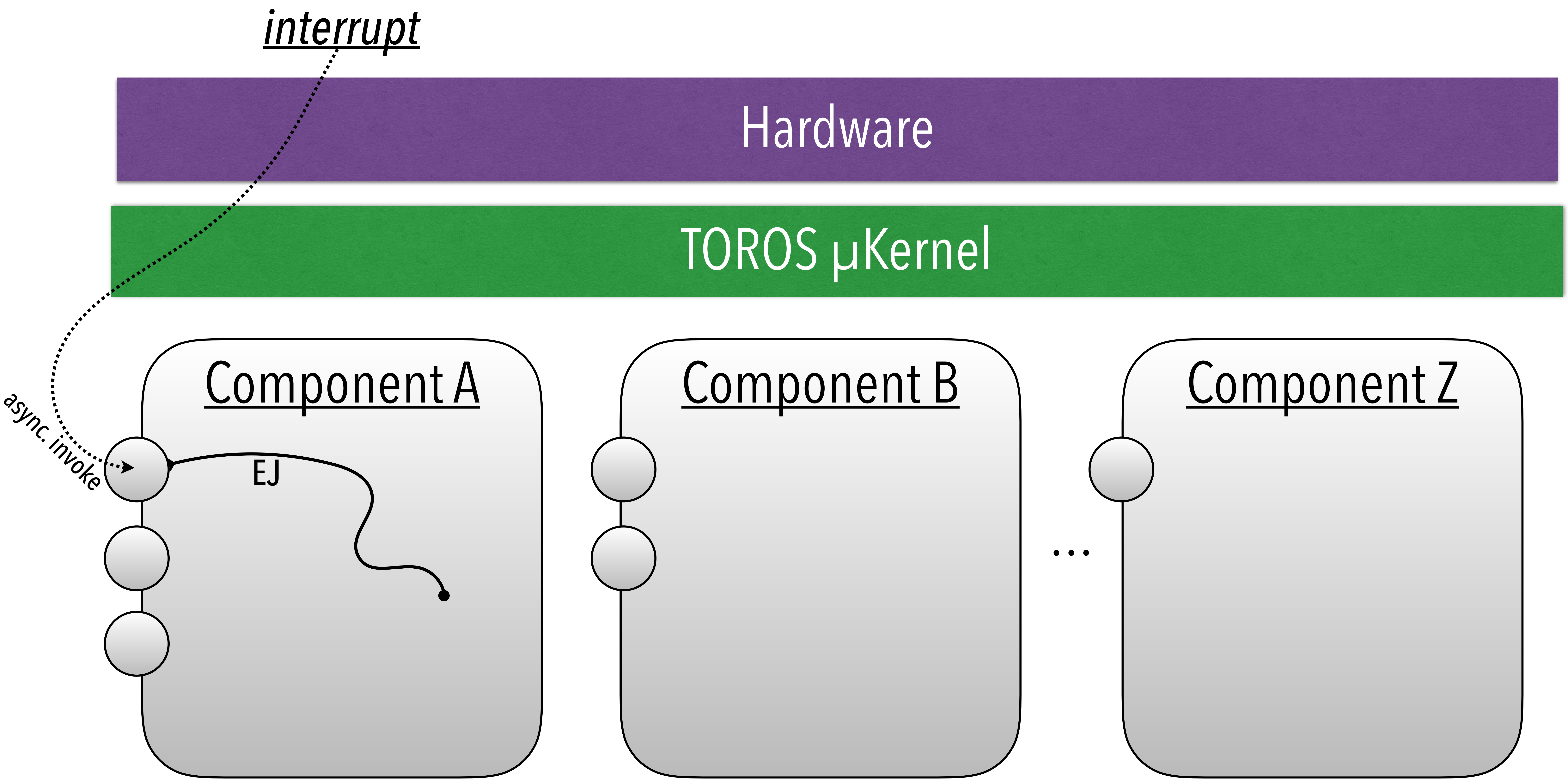
EXECUTION & PROGRAMMING MODEL



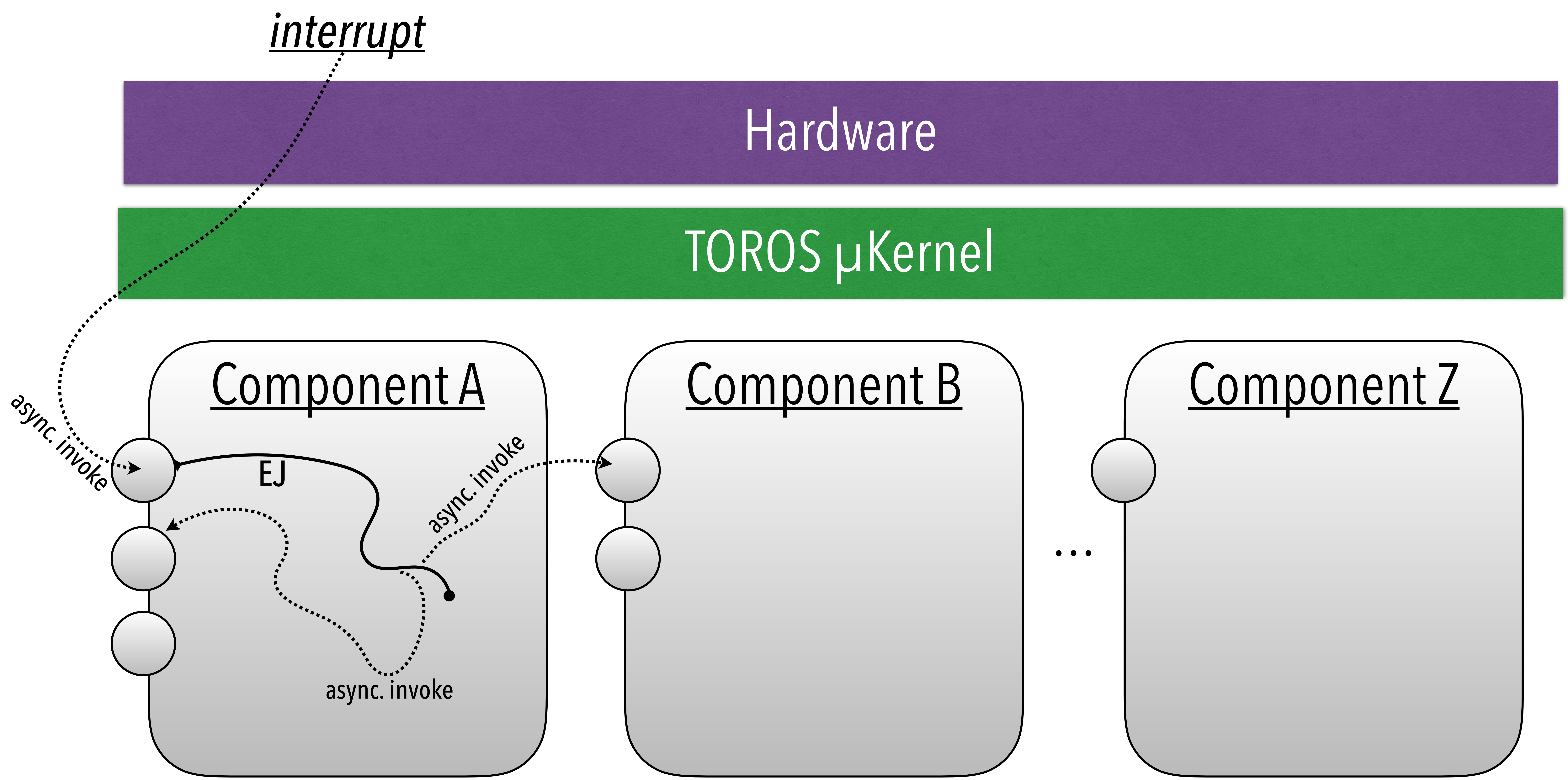
EXECUTION & PROGRAMMING MODEL



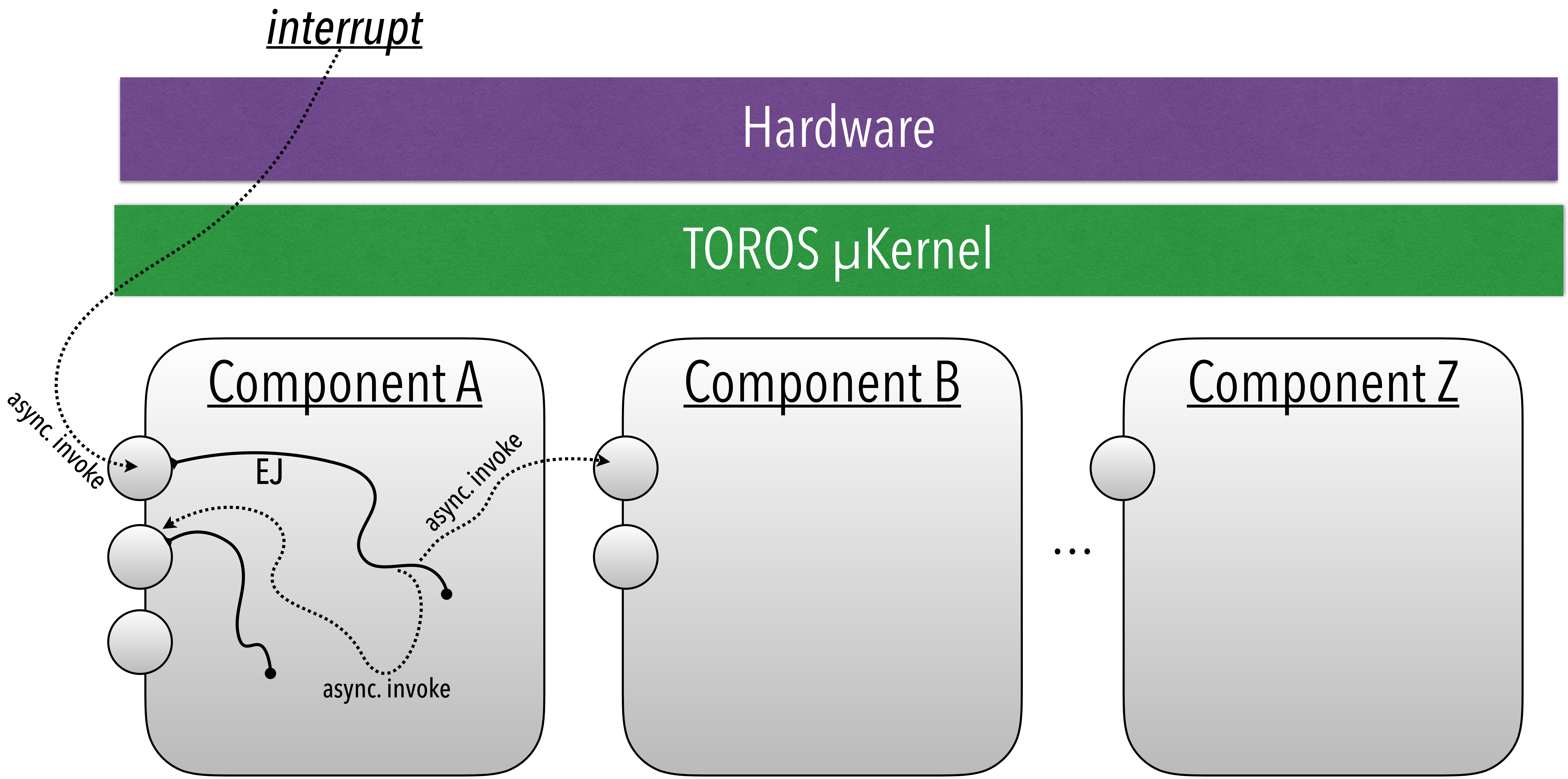
EXECUTION & PROGRAMMING MODEL



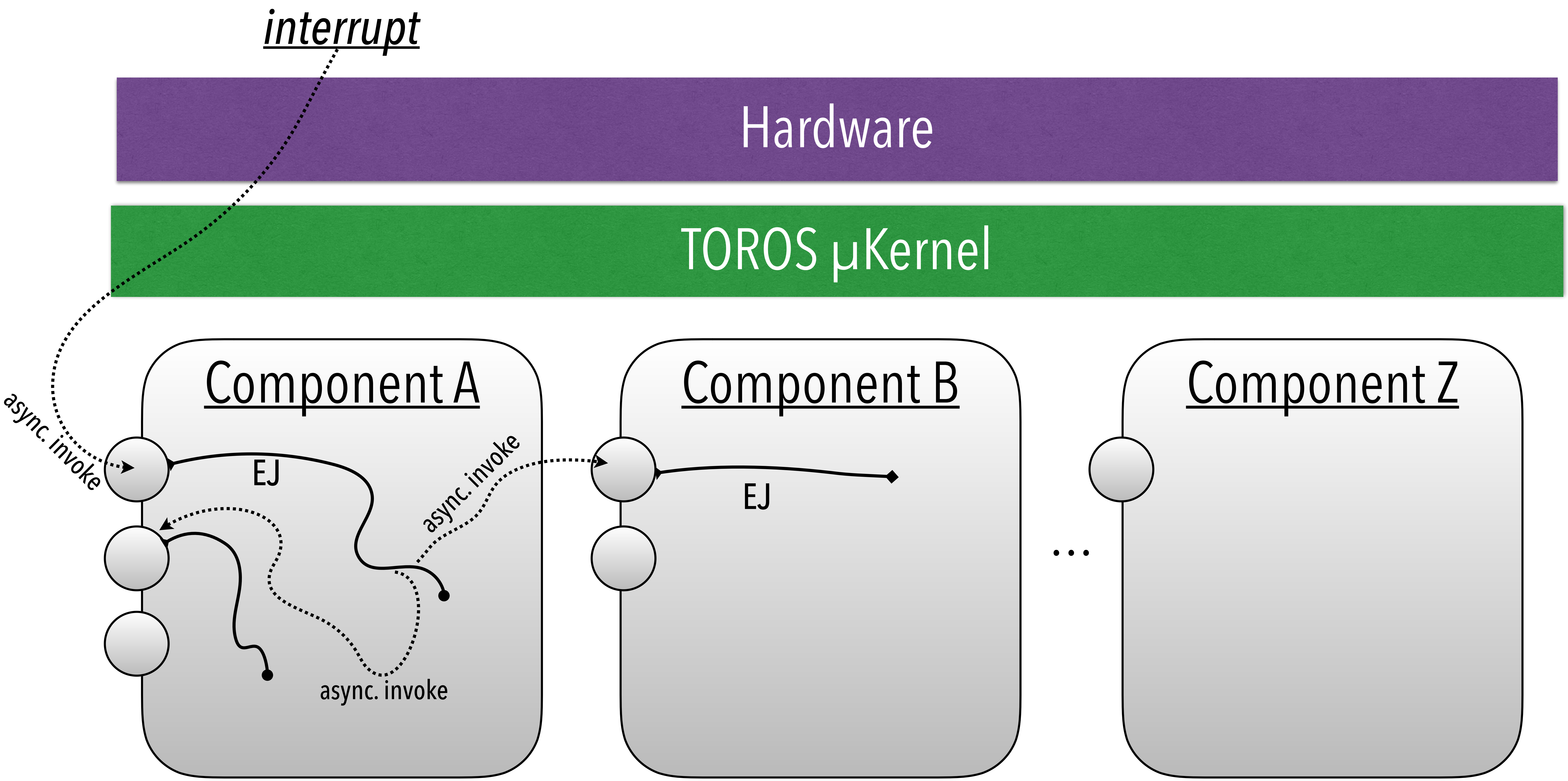
EXECUTION & PROGRAMMING MODEL



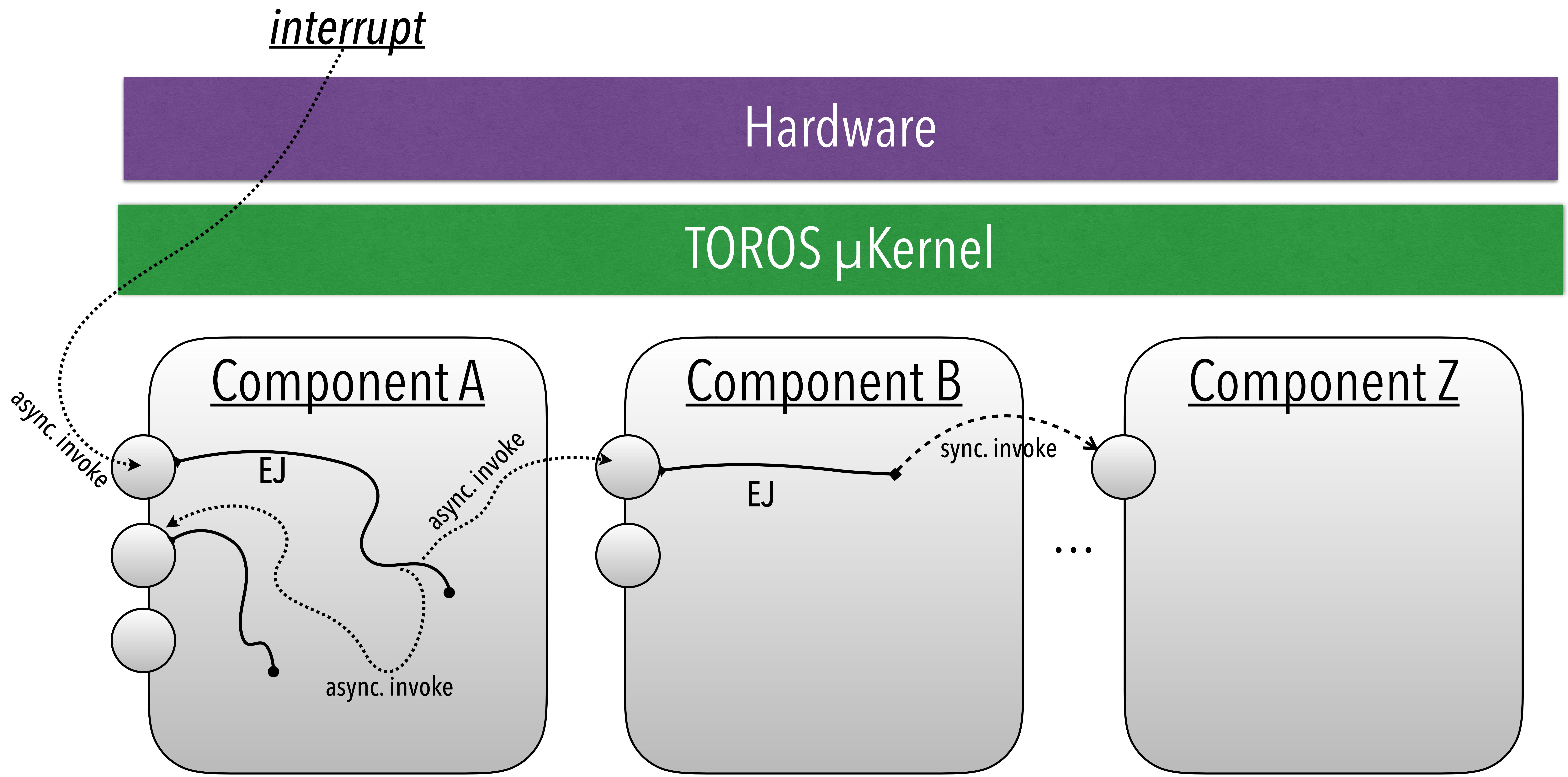
EXECUTION & PROGRAMMING MODEL



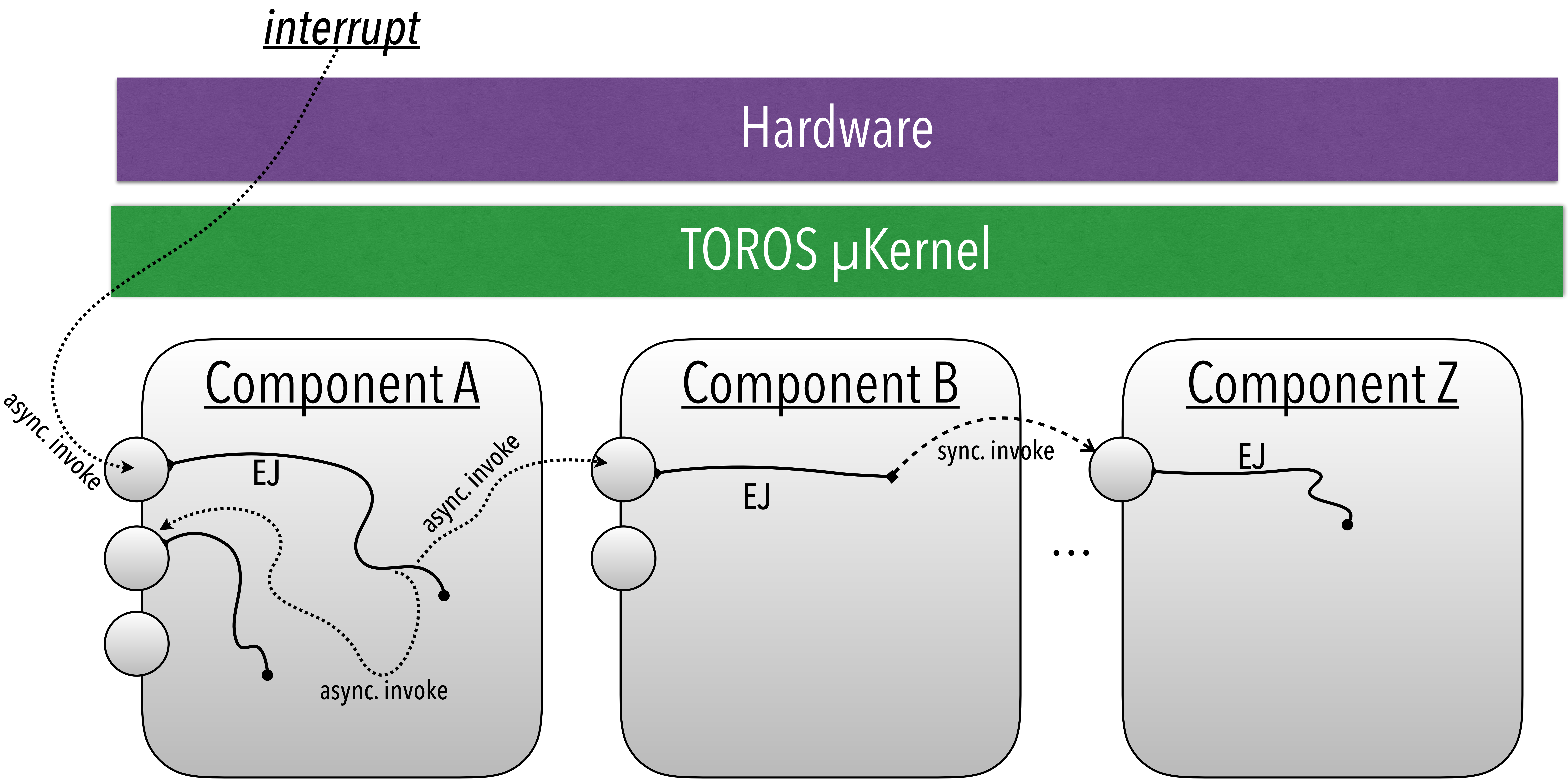
EXECUTION & PROGRAMMING MODEL



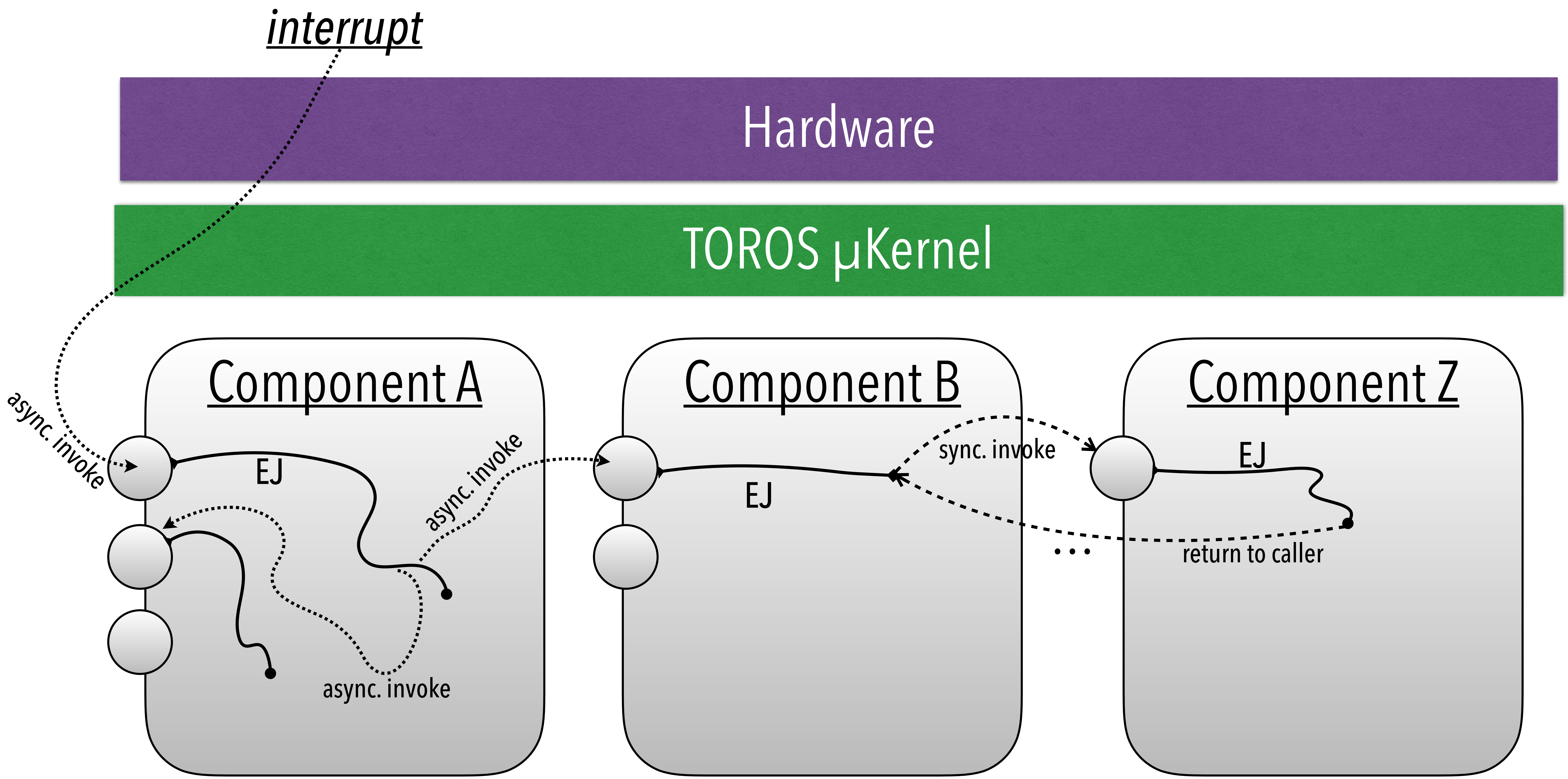
EXECUTION & PROGRAMMING MODEL



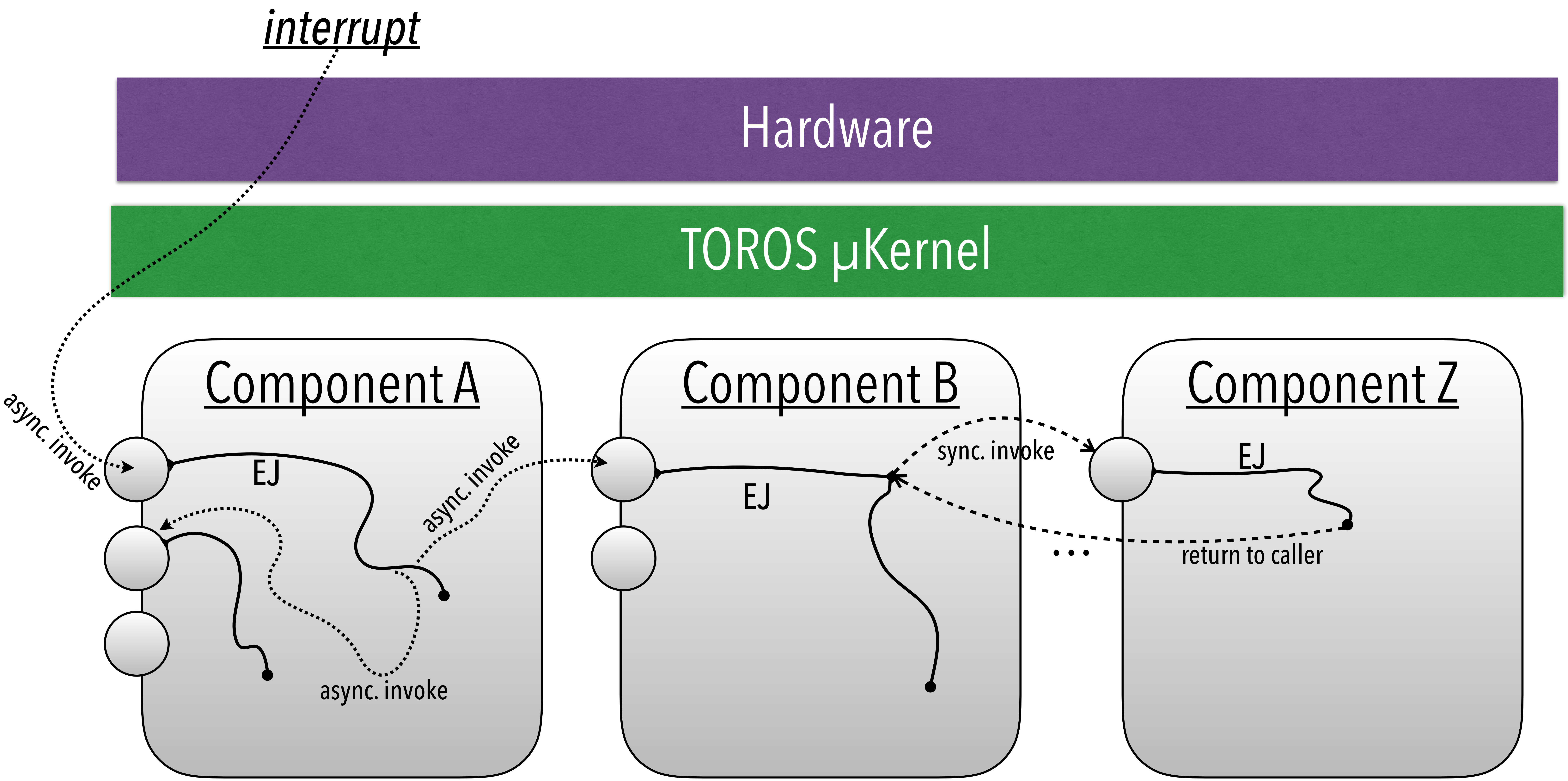
EXECUTION & PROGRAMMING MODEL



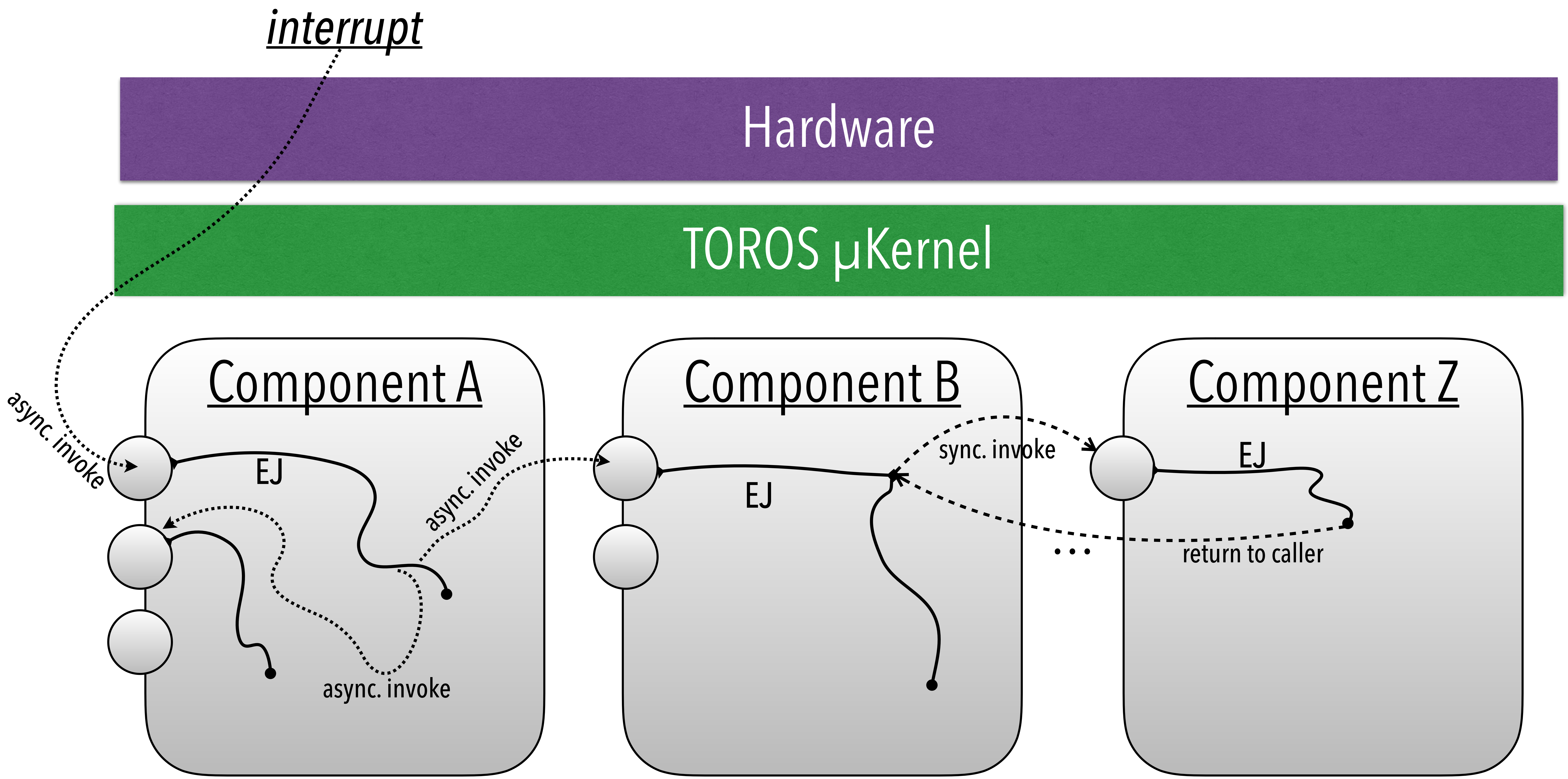
EXECUTION & PROGRAMMING MODEL



EXECUTION & PROGRAMMING MODEL



EXECUTION & PROGRAMMING MODEL



➔ event-driven / continuation-based / actor-like programming model

microkernel philosophy	vs	**opinionated design**
policy freedom		*freedom from choice*

<u>microkernel philosophy</u>

policy freedom

vs

<u>opinionated design</u>

freedom from choice

Aim for **maximal flexibility**: under no circumstance hardcode any policy into the OS.

The application developer knows best.

microkernel philosophy

policy freedom

vs

opinionated design

freedom from choice

Aim for **maximal flexibility**: under no circumstance hardcode any policy into the OS.

The application developer knows best.

Aim for **maximal simplicity**: hide as many choices as possible from the application developer.

"Trust me, I know what's best in this domain."

microkernel philosophy

policy freedom

vs

opinionated design

freedom from choice

Aim for **maximal flexibility**: under no circumstance hardcode any policy into the OS.

The application developer knows best.

Aim for **maximal simplicity**: hide as many choices as possible from the application developer.

"Trust me, I know what's best in this domain."

L4 & Composite — Linux, RTEMS, FreeRTOS… — TOROS

(3) TEMPORAL REFLECTION

transparently & continuously self-assess **temporal correctness**
and **proactively adapt** when guarantees can no longer be given

(3) TEMPORAL REFLECTION

transparently & continuously self-assess **temporal correctness** and **proactively adapt** when guarantees can no longer be given

The analysis is non-optional and not a separate tool.

(3) TEMPORAL REFLECTION

transparently & continuously self-assess **temporal correctness** and **proactively adapt** when guarantees can no longer be given

The analysis is non-optional and not a separate tool.

Approach

- ➔ always-on **lightweight tracing** of entire system
- ➔ trigger **incremental re-analysis** whenever inputs to analysis change
- ➔ invoke application-provided **adaptation handler** if timing goals cannot be guaranteed

(3) TEMPORAL REFLECTION

transparently & continuously self-assess **temporal correctness** and **proactively adapt** when guarantees can no longer be given

The analysis is non-optional and not a separate tool.

Approach

➔ always-on **lightweight tracing** of entire system

➔ trigger **incremental re-analysis** whenever inputs to analysis change

➔ invoke application-provided **adaptation handler** if timing goals cannot be guaranteed

Challenges

➔ tracing runtime overheads

➔ tracing space overheads

➔ analysis runtime

(3) TEMPORAL REFLECTION

transparently & continuously self-assess temporal correctness and proactively adapt when guarantees can no longer be given

The analysis is non-optional and not a separate tool.

Approach

- ➔ always-on lightweight tracing of entire system
- ➔ trigger incremental re-analysis whenever inputs to analysis change
- ➔ invoke application-provided adaptation handler if timing goals cannot be guaranteed

Challenges

- ➔ tracing runtime overheads
- ➔ tracing space overheads
- ➔ analysis runtime

Plan B

- ➔ cloud offloading of analysis

(3) TEMPORAL REFLECTION

transparently & continuously self-assess temporal correctness and proactively adapt when guarantees can no longer be given

Do not measure WCETs!

We can trace percentiles (< 100) and (non-)correlations with high & quantifiable confidence in bounded space.

Approach

- ➔ always-on lightweight tracing of entire system
- ➔ trigger incremental re-analysis whenever inputs to analysis change
- ➔ invoke application-provided adaptation handler if timing goals cannot be guaranteed

Challenges

- ➔ tracing runtime overheads
- ➔ tracing space overheads
- ➔ analysis runtime

Plan B

- ➔ cloud offloading of analysis

(4) STRUCTURED UNCERTAINTY MANAGEMENT

provide **first-class, sound abstractions** to manage uncertainty due to **below-worst-case provisioning**

(4) STRUCTURED UNCERTAINTY MANAGEMENT

provide **first-class, sound abstractions** to manage uncertainty
due to **below-worst-case provisioning**

Slack Reclamation + Slack Pools
Reallocate unused surplus budget of one activity to another one in need.

(4) STRUCTURED UNCERTAINTY MANAGEMENT

provide **first-class, sound abstractions** to manage uncertainty due to **below-worst-case provisioning**

Slack Reclamation + Slack Pools
Reallocate unused surplus budget of one activity to another one in need.

Correlation-aware probabilistic sensitivity analysis w/o WCETs

(4) STRUCTURED UNCERTAINTY MANAGEMENT

provide **first-class, sound abstractions** to manage uncertainty due to **below-worst-case provisioning**

<u>Slack Reclamation + Slack Pools</u>
Reallocate unused surplus budget of one activity to another one in need.

Correlation-aware probabilistic sensitivity analysis w/o WCETs

<u>Expected Safety Margin</u>
Bound the amount of slack available in the expected case.

(4) STRUCTURED UNCERTAINTY MANAGEMENT

provide **first-class, sound abstractions** to manage uncertainty due to **below-worst-case provisioning**

Slack Reclamation + Slack Pools
Reallocate unused surplus budget of one activity to another one in need.

Correlation-aware probabilistic sensitivity analysis w/o WCETs

Expected Safety Margin
Bound the amount of slack available in the expected case.

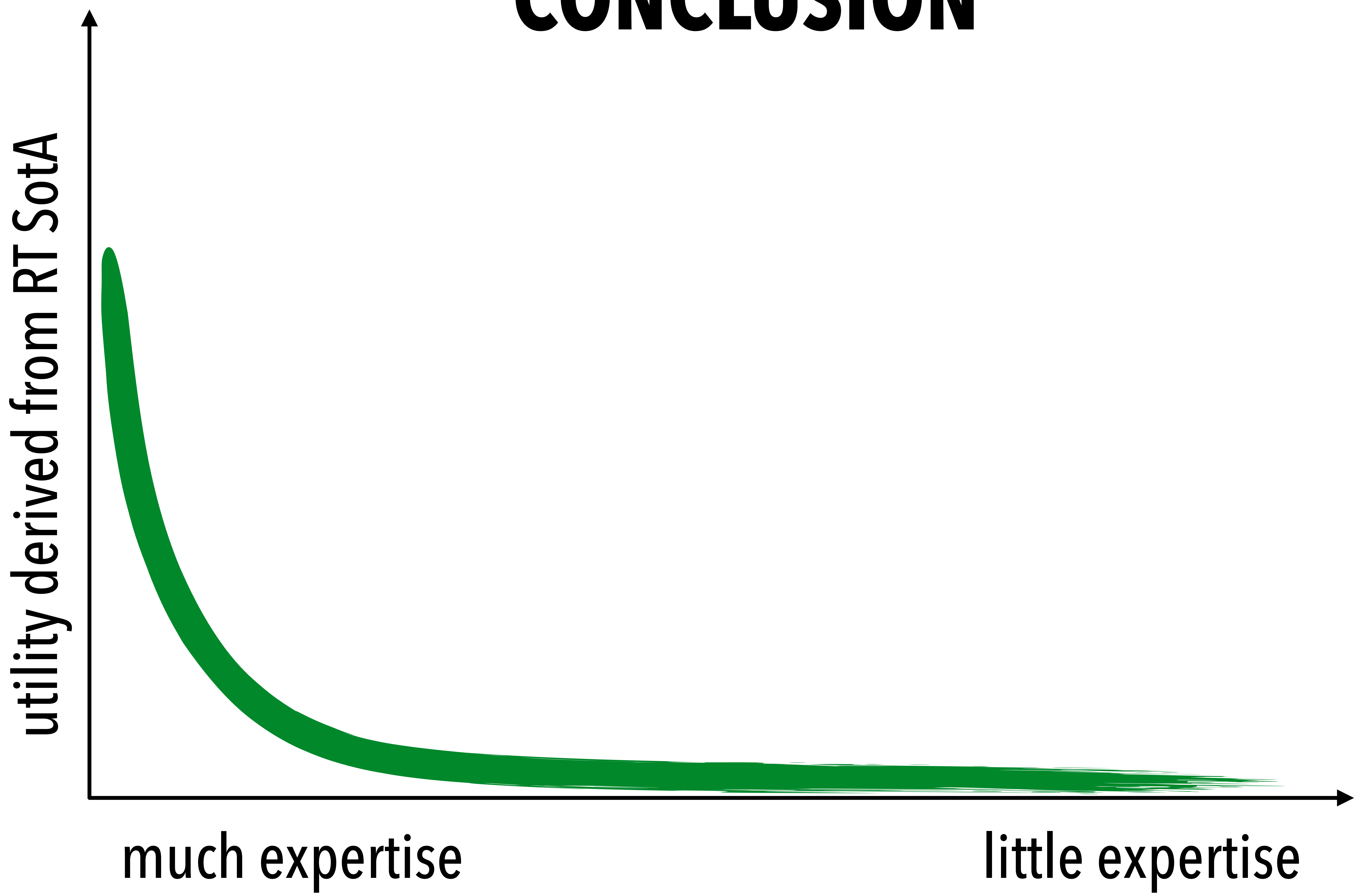
Desired Guarantee: "Margin to Cliff" instead of "Yes/No"

"an increase in execution time by X% has no ill effects with probability at least Y"

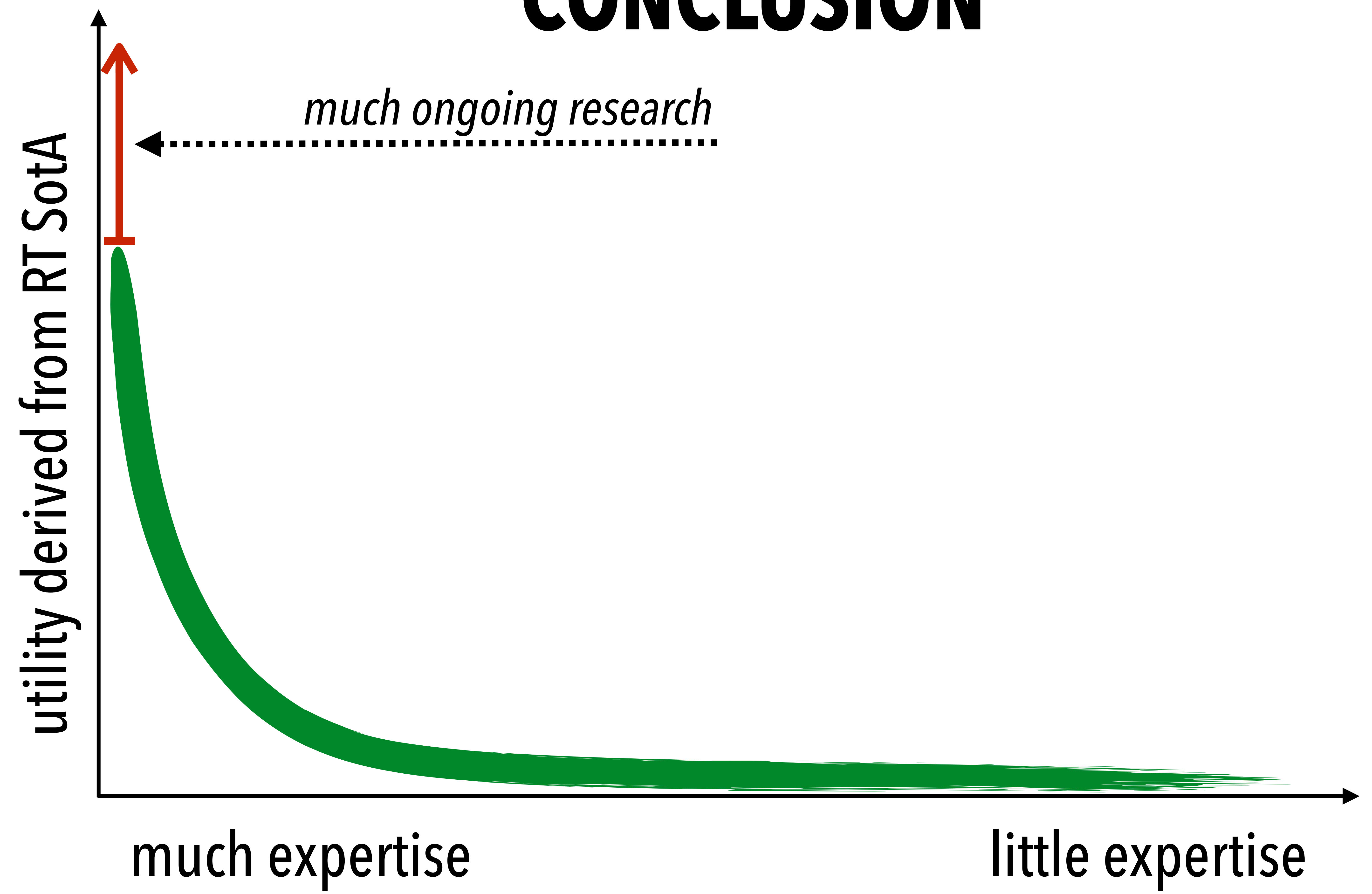
CONCLUSION

European Research Council
Established by the European Commission

CONCLUSION

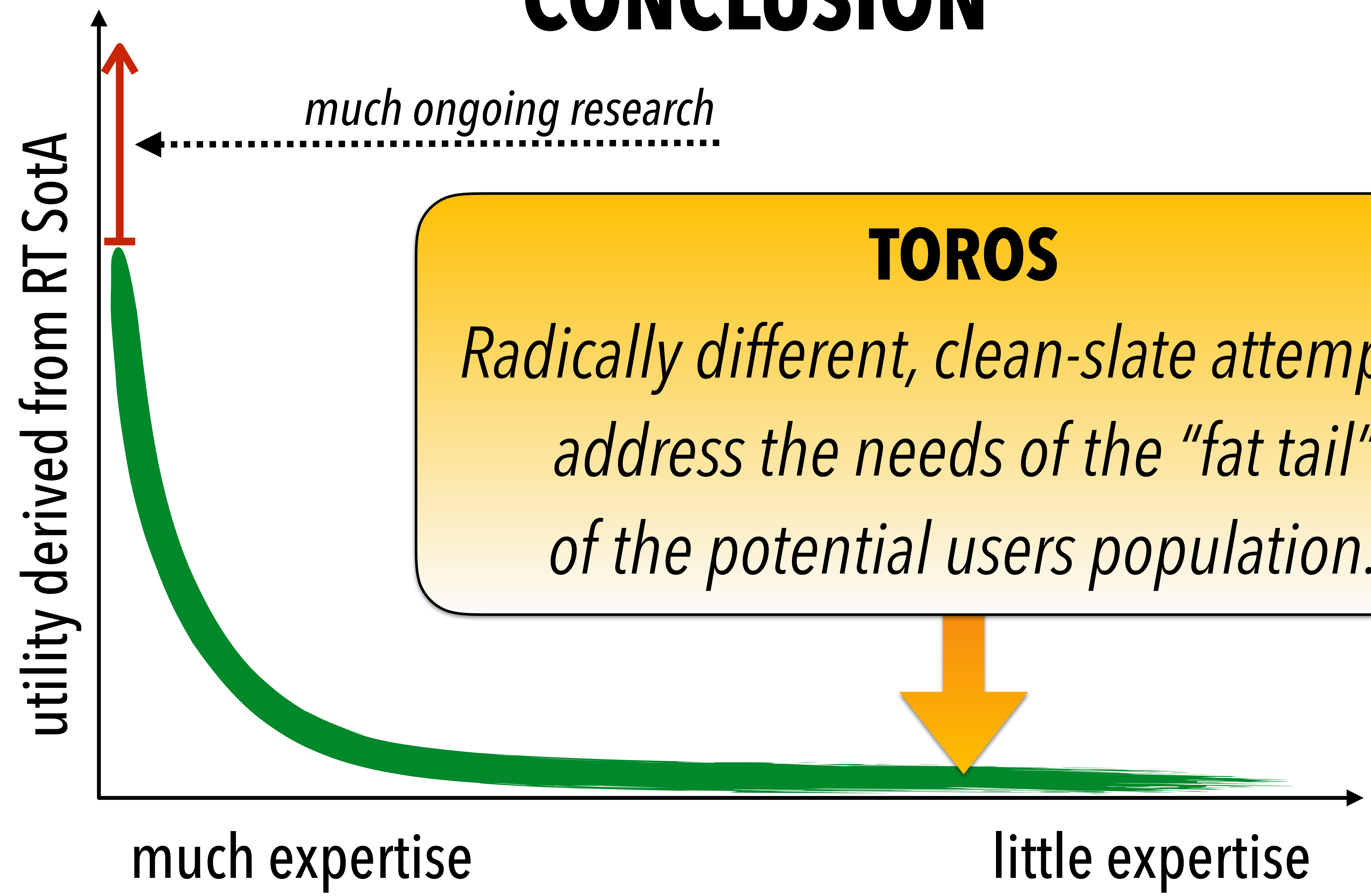

utility derived from RT SotA
much expertise
little expertise

CONCLUSION



European Research Council
Established by the European Commission

CONCLUSION

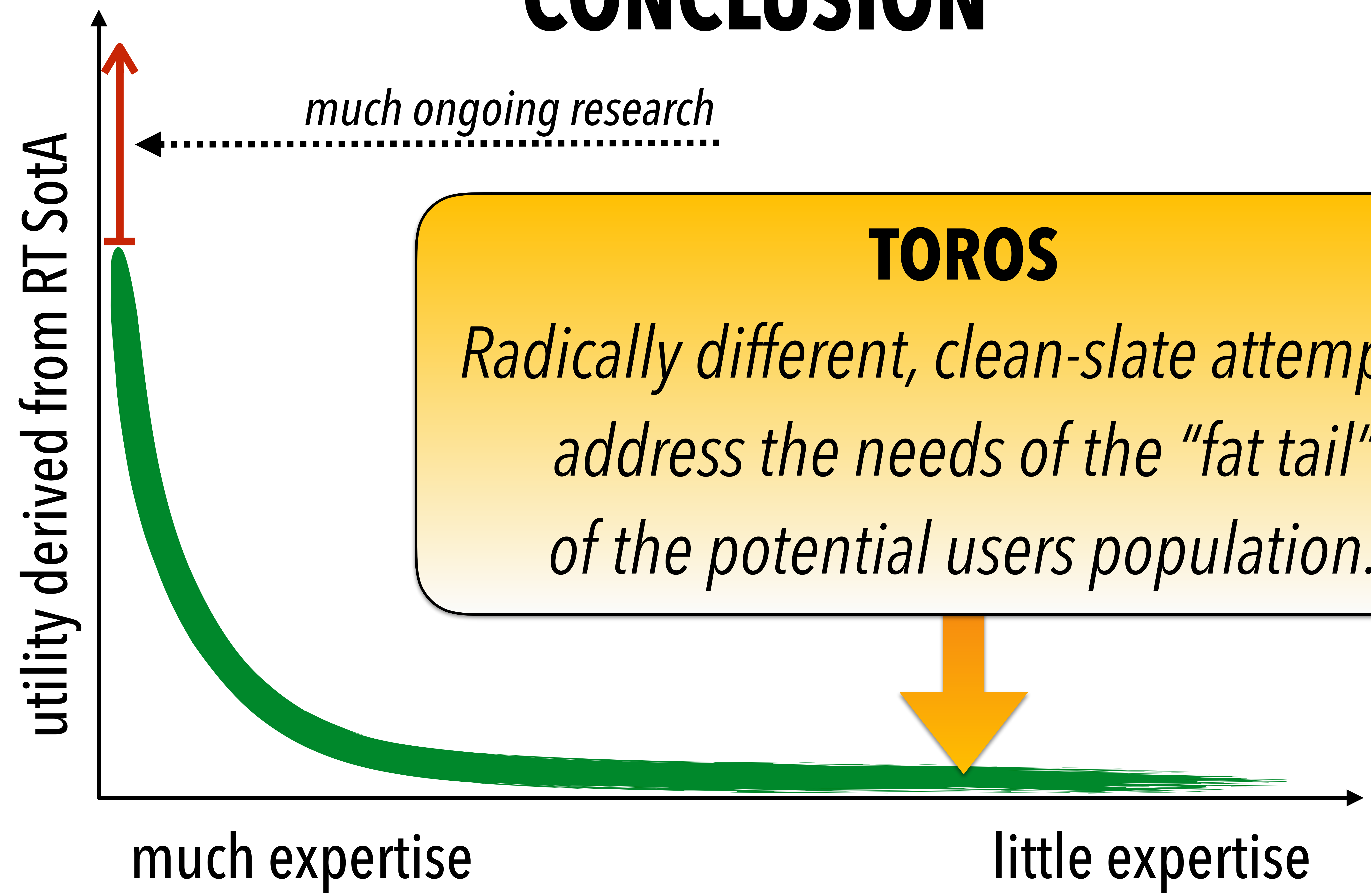


TOROS

Radically different, clean-slate attempt to address the needs of the "fat tail" of the potential users population.

European Research Council
Established by the European Commission

CONCLUSION



TOROS

Radically different, clean-slate attempt to address the needs of the "fat tail" of the potential users population.

The Five TOROS Principles

1. Theory-Oriented RTOS Design
2. Declarative OS Abstractions
3. Temporal Reflection
4. Structured Uncertainty Management
5. Trustworthy Analysis

European Research Council
Established by the European Commission