Provable Security

Henrik Karlsson (henrik10@kth.se)

Supervisors: Mads Dam & Roberto Guanciale
KTH Royal Institute of Technology

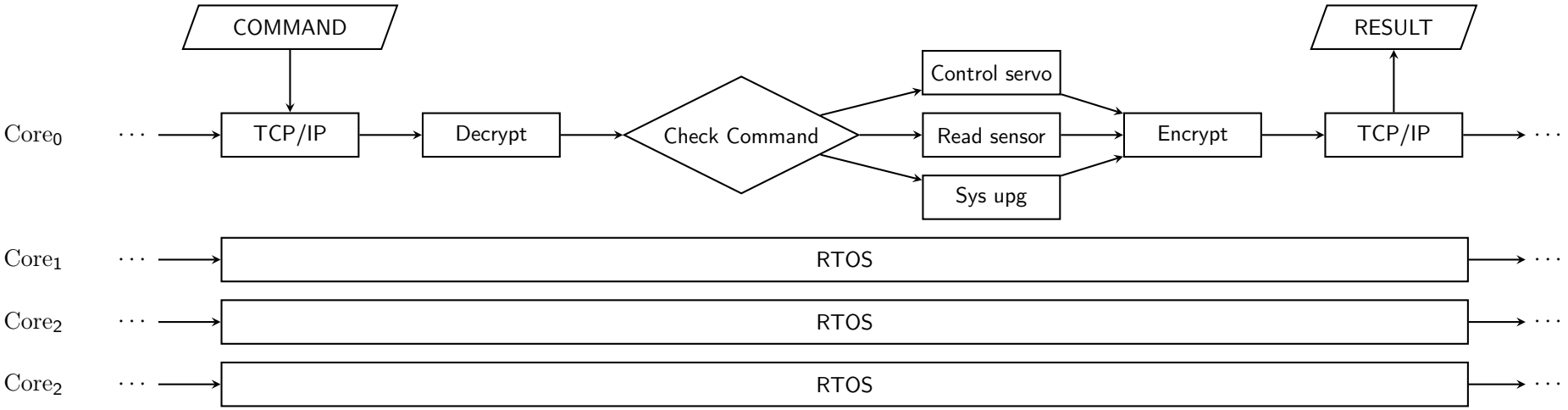
October 27, 2022

Outline

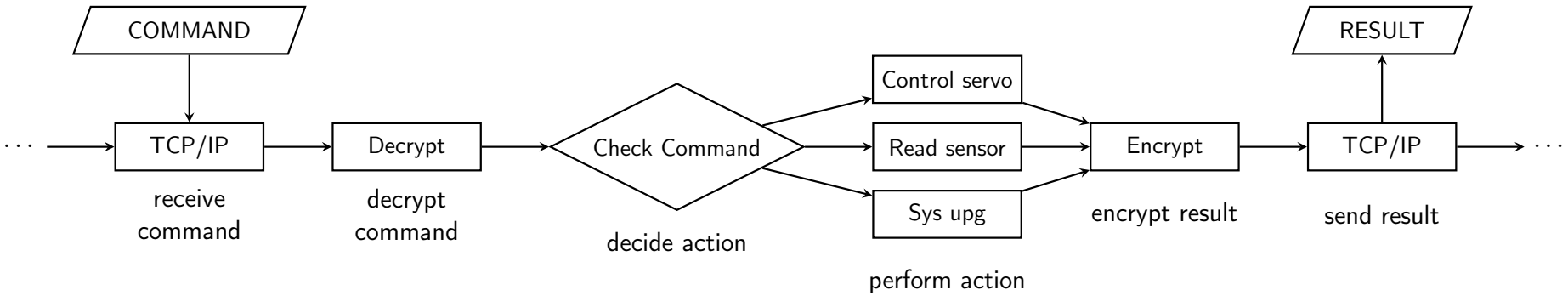
1. Example Application & Proof-of-Concept
2. Simply Secure Separation Kernel (S3K)
3. S3K Process Scheduling
4. Multicore HolBA & Kernel Verification
5. Research plan

Example Application & Proof-of-Concept

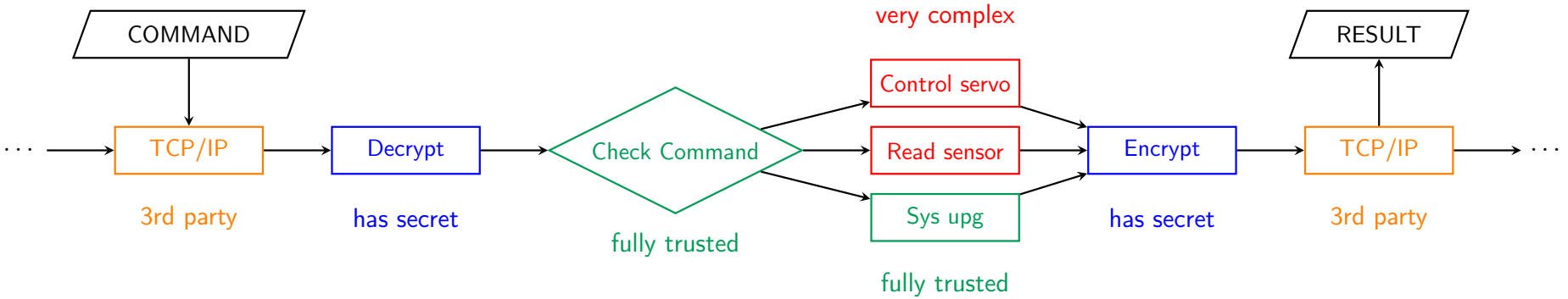
Application



Application

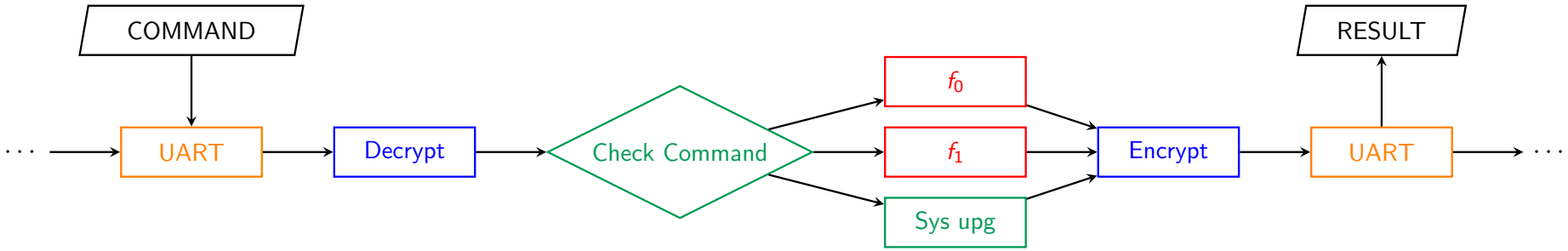


Application



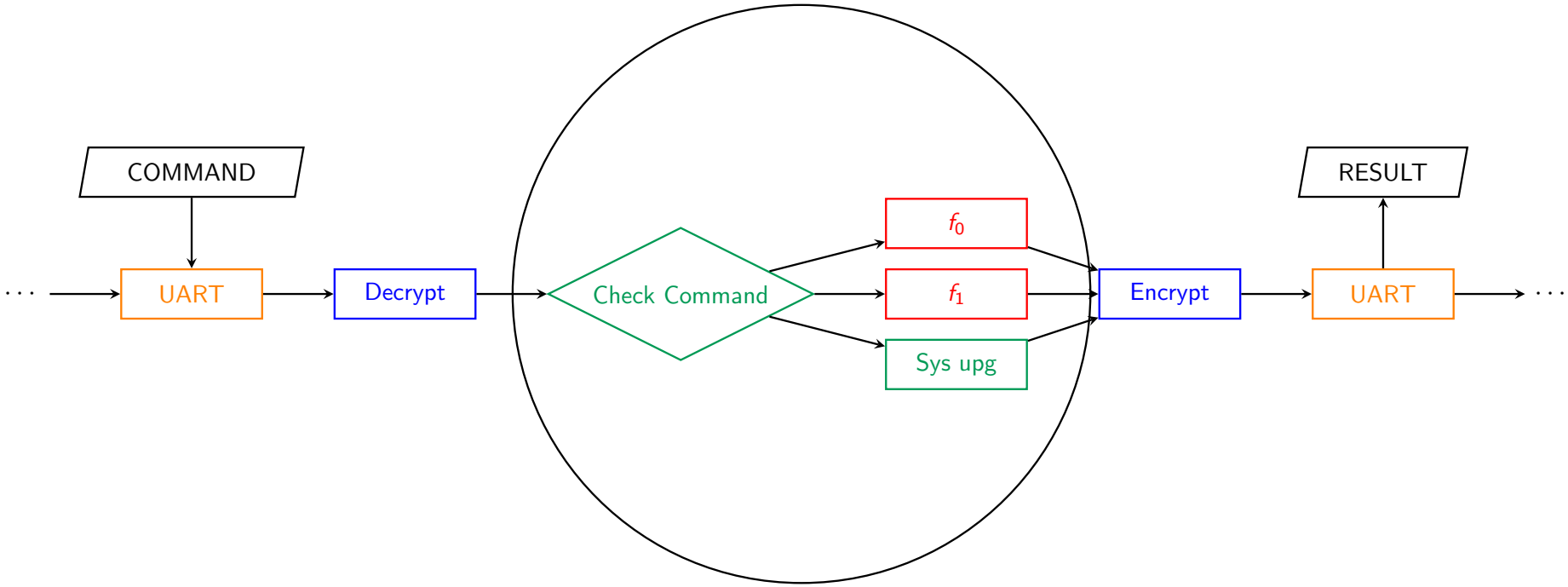
- Monitor – Monitors system & handles commands
- Functions – Interact with environment and output data
- TCP/IP – Handle TCP/IP communication
- Crypto engine – Encrypts & Decrypts Packages

Proof-of-Concept



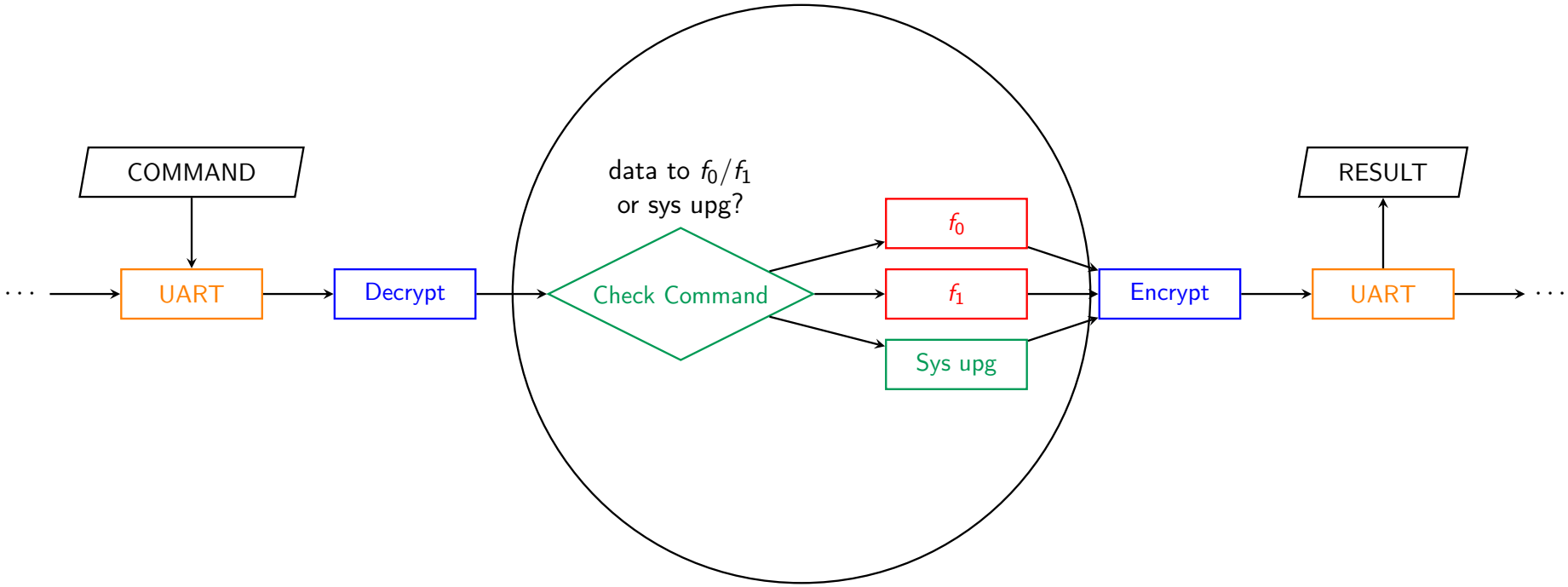
- Monitor – Manage apps & handles commands
- Dummy Functions f_0, f_1 – Process and output data
- UART – Handle UART communication
- Crypto engine – Encrypts & Decrypts Packages

Proof-of-Concept



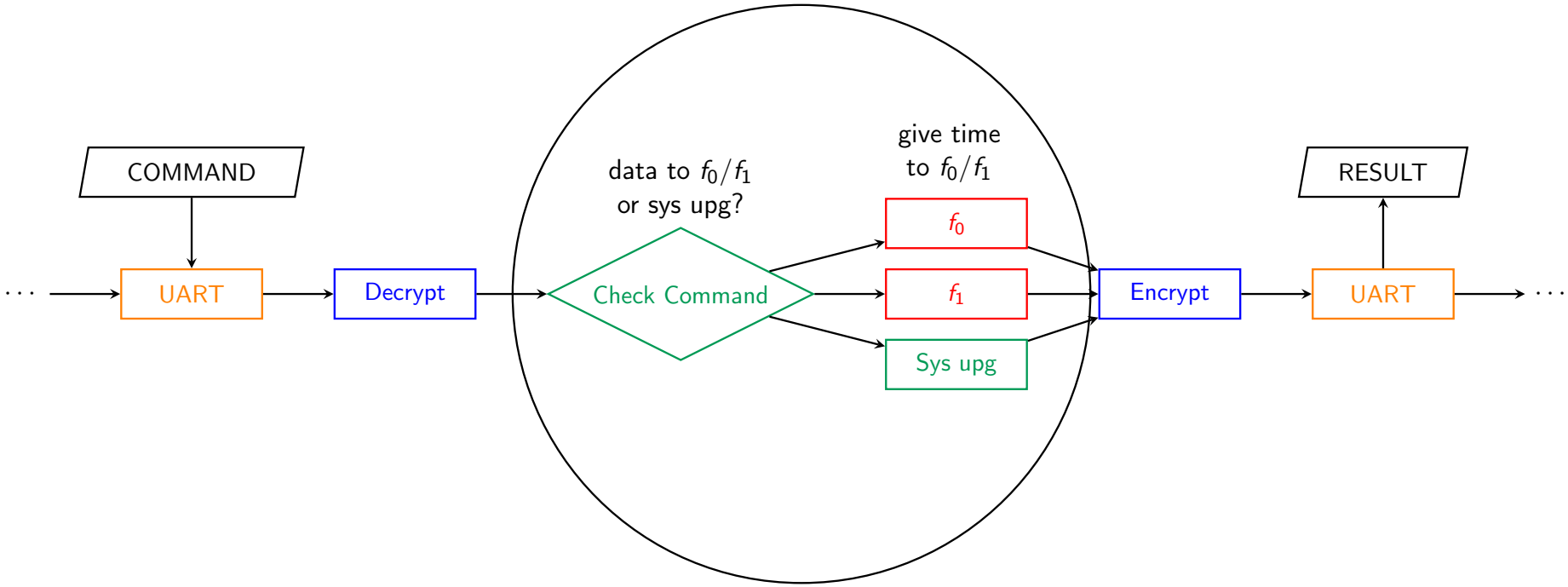
- Monitor – Manage apps & handles commands
- Dummy Functions f_0, f_1 – Process and output data
- UART – Handle UART communication
- Crypto engine – Encrypts & Decrypts Packages

Proof-of-Concept



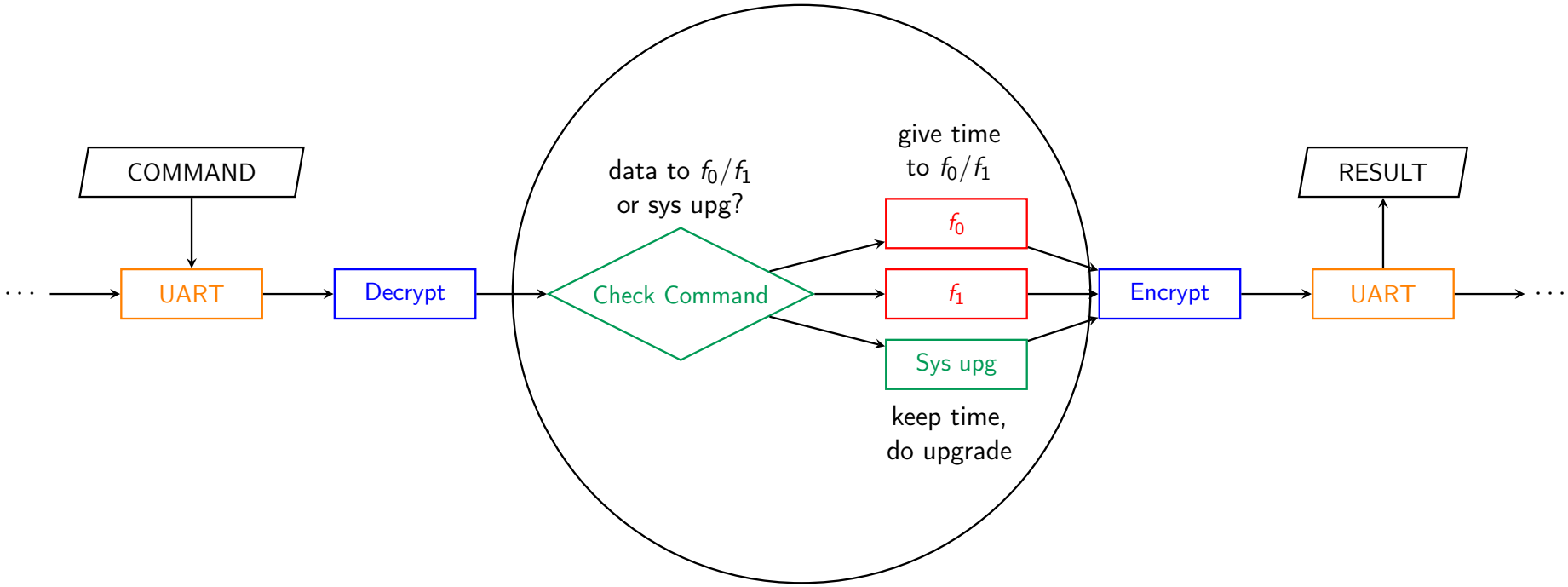
- Monitor – Manage apps & handles commands
- Dummy Functions f_0, f_1 – Process and output data
- UART – Handle UART communication
- Crypto engine – Encrypts & Decrypts Packages

Proof-of-Concept



- Monitor – Manage apps & handles commands
- Dummy Functions f_0, f_1 – Process and output data
- UART – Handle UART communication
- Crypto engine – Encrypts & Decrypts Packages

Proof-of-Concept



- Monitor – Manage apps & handles commands
- Dummy Functions f_0, f_1 – Process and output data
- UART – Handle UART communication
- Crypto engine – Encrypts & Decrypts Packages

Simply Secure Separation Kernel (S3K)

Simply Secure Separation Kernel (S3K)

- Designed and implemented a separation kernel with . . .

[1] MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)

- Designed and implemented a separation kernel with . . .
  - Memory Protection and Management

[1] MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)

- Designed and implemented a separation kernel with . . .
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems

[1] MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)

- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)

[1] MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)

- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring

[1] MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)

- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring
- Targeting standard RISC-V 64-bit (RV64IMA) with MPU[1]

[1] MPU - Memory Protection Unit, protects physical memory.

Simply Secure Separation Kernel (S3K)

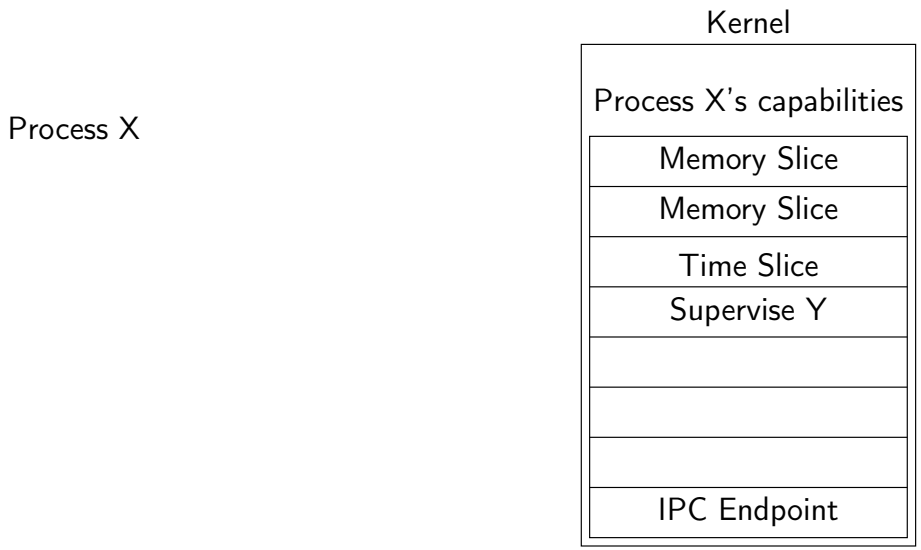
- Designed and implemented a separation kernel with ...
  - Memory Protection and Management
  - Secure Time Management for Real-Time Systems
  - Secure Inter-Process Communication (IPC)
  - Process Monitoring
- Targeting standard RISC-V 64-bit (RV64IMA) with MPU[1]
- $\sim$ 2000 lines of C/Assembly

[1] MPU - Memory Protection Unit, protects physical memory.

Capabilities

Previously mentioned features are implemented using capabilities

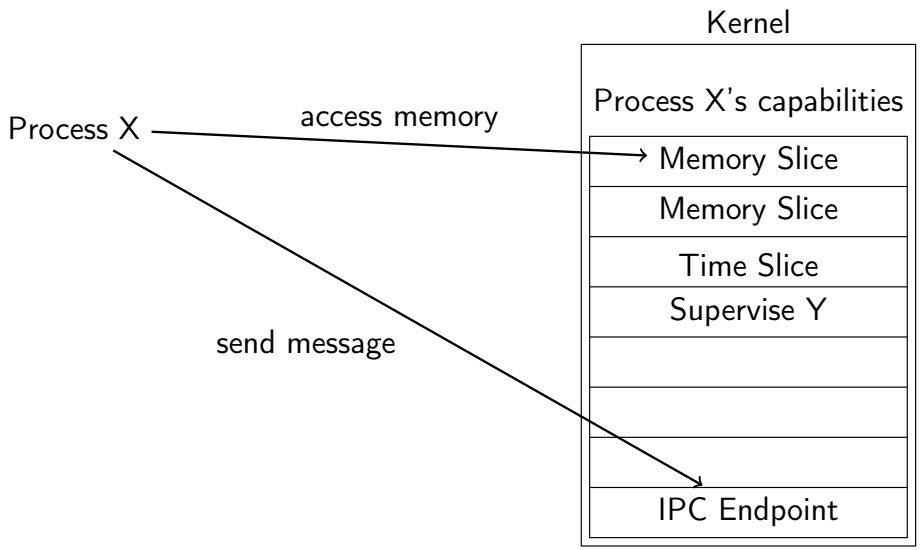
- Capability = object in kernel representing a resource



Capabilities

Previously mentioned features are implemented using capabilities

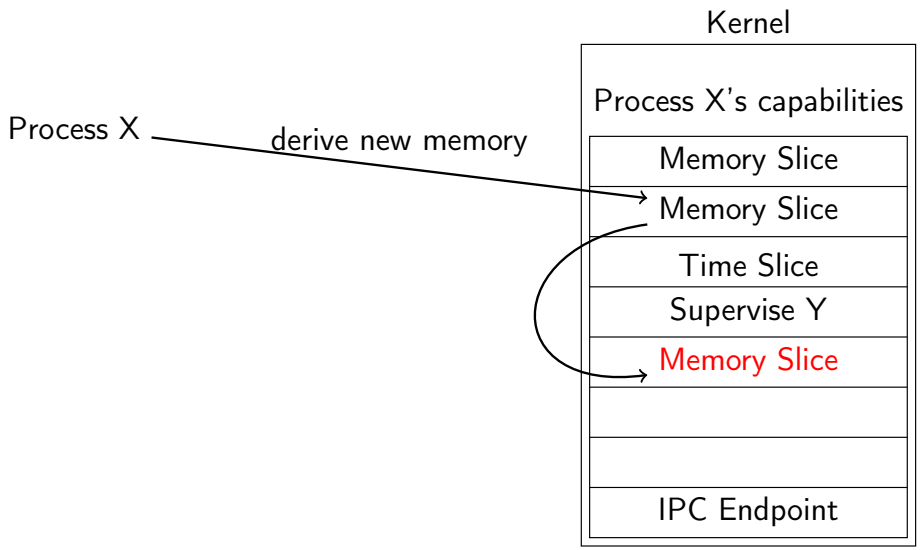
- Capability = object in kernel representing a resource
- Process owning a capability has access to corresponding resource



Capabilities

Previously mentioned features are implemented using capabilities

- Capability = object in kernel representing a resource
- Process owning a capability has access to corresponding resource
- Process can derive new capabilities from existing capabilities



S3K Capabilities

- Memory Slice – Manage access to a physical memory region.
  - PMP – Configure RISC-V's MPU, grants memory access.

S3K Capabilities

- Memory Slice – Manage access to a physical memory region.
  - PMP – Configure RISC-V's MPU, grants memory access.
- Time Slice – Manage and grant execution time on a core.

S3K Capabilities

- Memory Slice – Manage access to a physical memory region.
  - PMP – Configure RISC-V's MPU, grants memory access.
- Time Slice – Manage and grant execution time on a core.
- Channels – Manage IPC channels and endpoints.
  - Receiver/Sender – Unidirectional IPC channel.
  - Server/Client – Bidirectional IPC channel.

S3K Capabilities

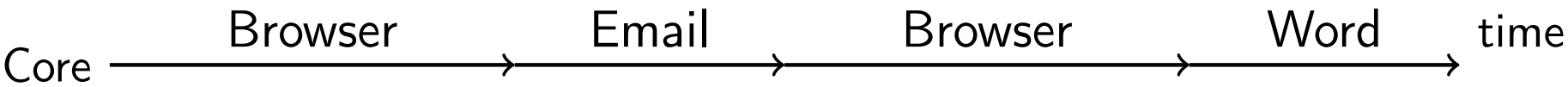
- Memory Slice – Manage access to a physical memory region.
  - PMP – Configure RISC-V's MPU, grants memory access.
- Time Slice – Manage and grant execution time on a core.
- Channels – Manage IPC channels and endpoints.
  - Receiver/Sender – Unidirectional IPC channel.
  - Server/Client – Bidirectional IPC channel.
- Supervisor – Manage a set of processes.

S3K Process Scheduling

What is Process Scheduling?

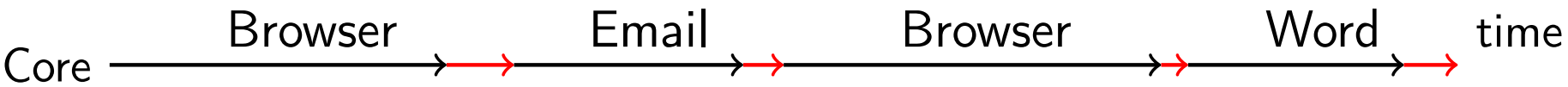
- Cores can only run one process at the time.

What is Process Scheduling?



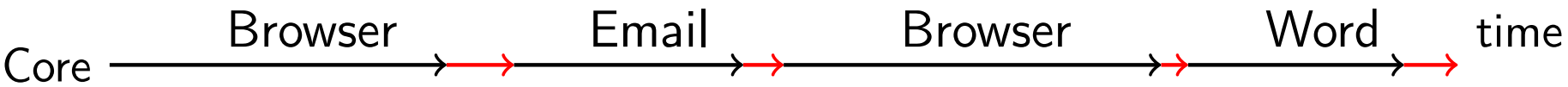
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.

What is Process Scheduling?



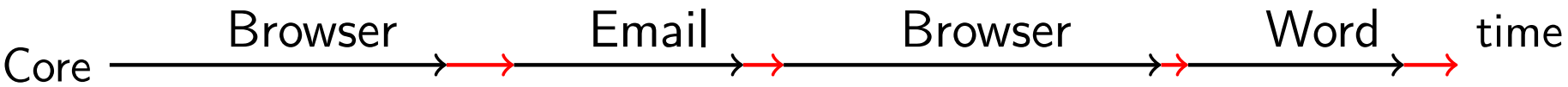
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.
- Context switch $\Longrightarrow$ overhead.

What is Process Scheduling?



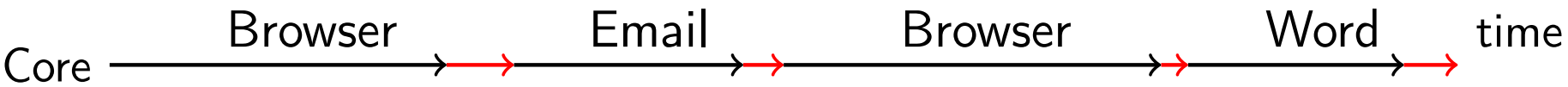
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.
- Context switch $\Longrightarrow$ overhead.
- Core multiplexing technique = Process Scheduling.

What is Process Scheduling?



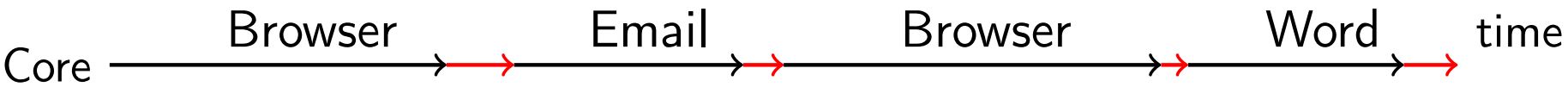
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.
- Context switch $\Longrightarrow$ overhead.
- Core multiplexing technique = Process Scheduling.
- Best scheduler? Performance vs. Safety vs. Security.

What is Process Scheduling?



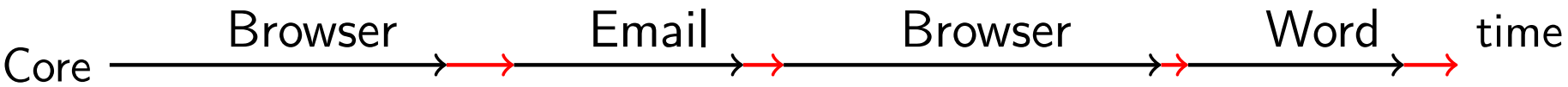
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.
- Context switch $\Longrightarrow$ overhead.
- Core multiplexing technique = Process Scheduling.
- Best scheduler? Performance vs. Safety vs. Security.
  - Processor utilization? Supercomputer, Desktop, ...

What is Process Scheduling?



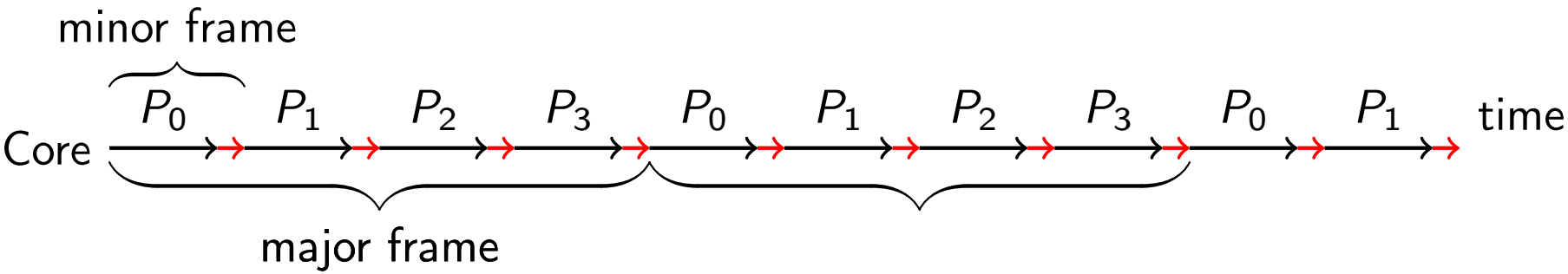
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.
- Context switch $\Longrightarrow$ overhead.
- Core multiplexing technique = Process Scheduling.
- Best scheduler? Performance vs. Safety vs. Security.
  - Processor utilization? Supercomputer, Desktop, ...
  - Meet deadline? Airplane, Car, Routers, Industrial Machines, ...

What is Process Scheduling?



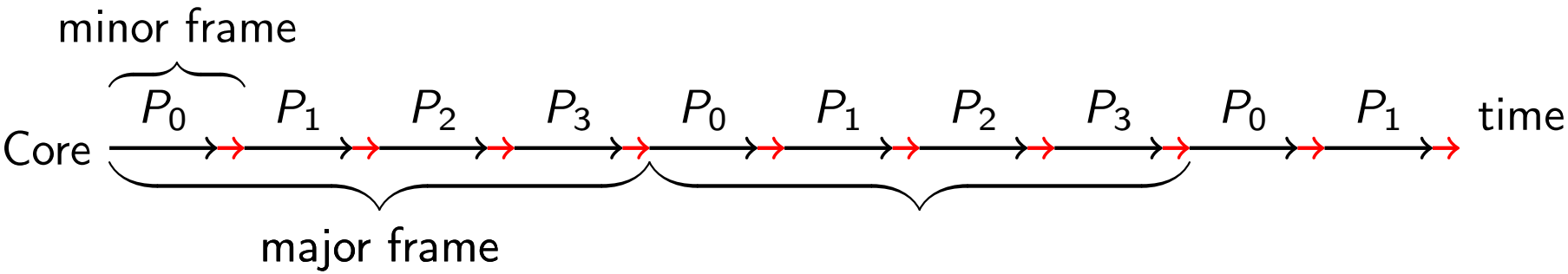
- Cores can only run one process at the time.
- Multiplexing the Core – main duty of the kernel.
- Context switch $\implies$ overhead.
- Core multiplexing technique = Process Scheduling.
- Best scheduler? Performance vs. Safety vs. Security.
  - Processor utilization? Supercomputer, Desktop, ...
  - Meet deadline? Airplane, Car, Routers, Industrial Machines, ...
  - Prevent side-channels attacks? Secure servers, routers, ...

S3K Scheduling



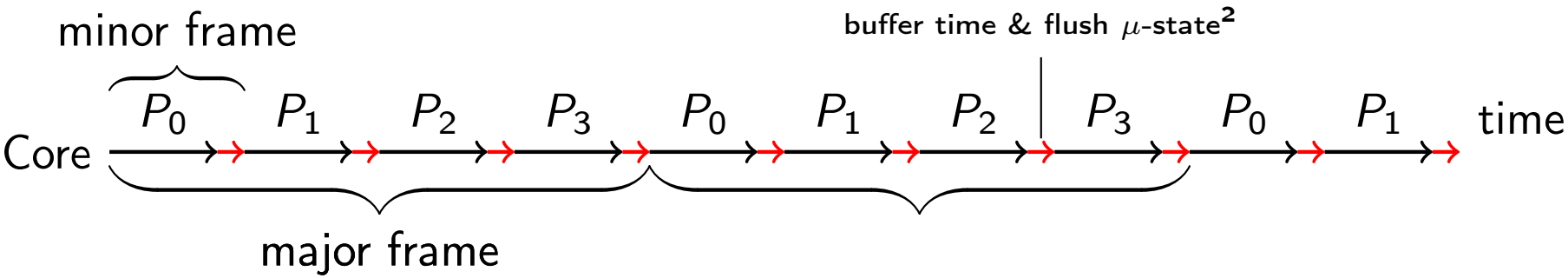
- Modified RR – Minor frames defined by time slice capabilities.

S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.

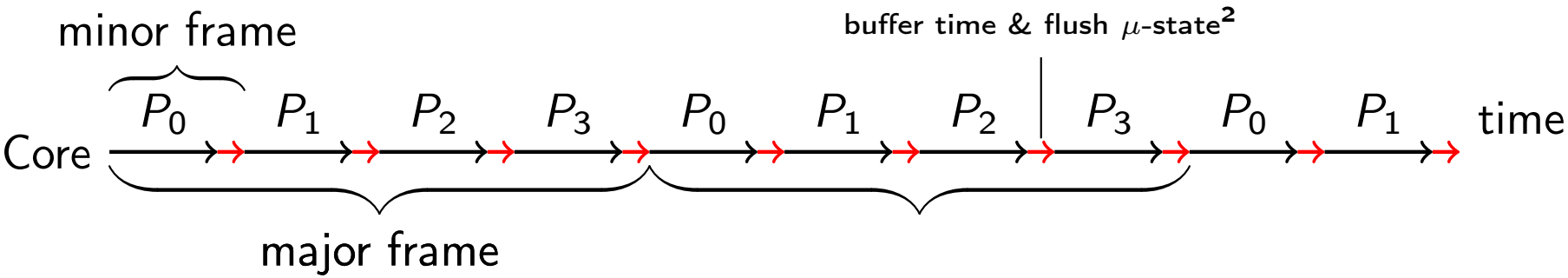
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.

[2] Flush cache, branch predictors, etc., support is hardware dependant.

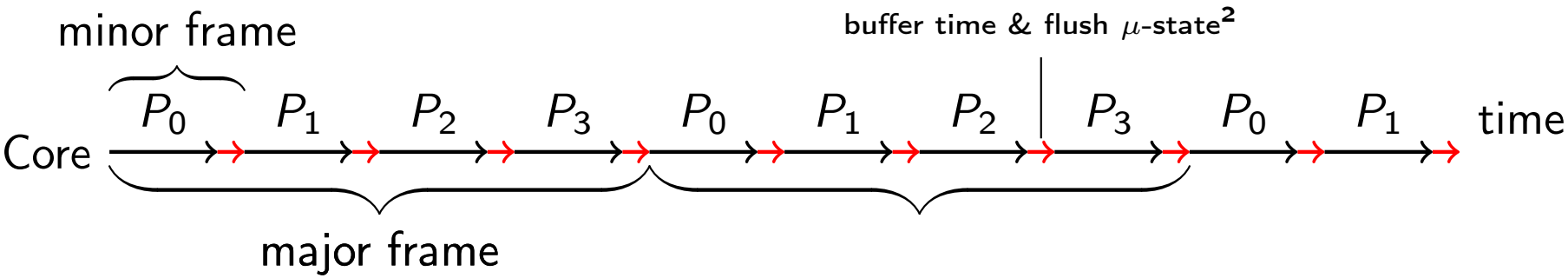
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.
- Temporal Isolation – A process's execution time depends only on its capabilities.

[2]Flush cache, branch predictors, etc., support is hardware dependant.

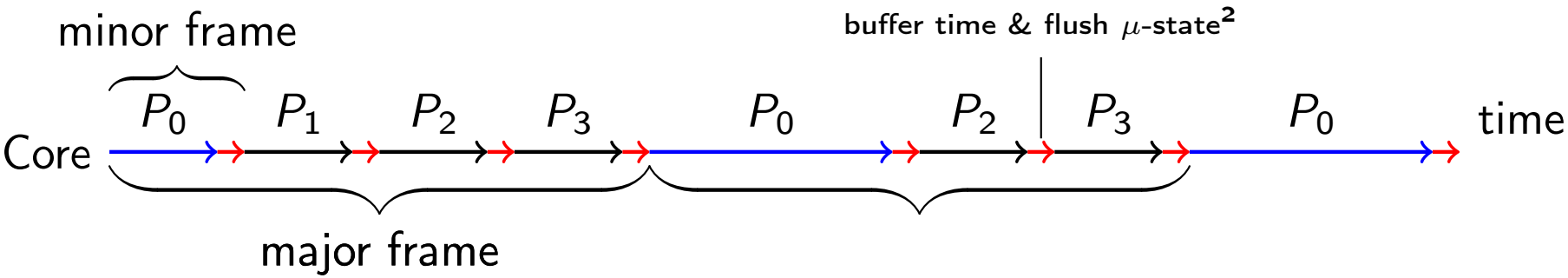
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.
- Temporal Isolation – A process's execution time depends only on its capabilities.
- Low-overhead – Scheduling decided by a lookup table.

[2]Flush cache, branch predictors, etc., support is hardware dependant.

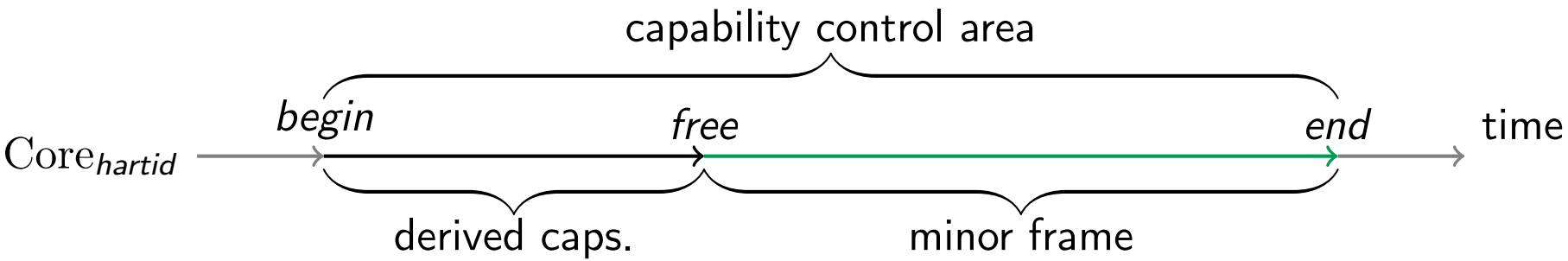
S3K Scheduling



- Modified RR – Minor frames defined by time slice capabilities.
- Fair – Process with execution time gets execution time.
- Predictable – Process knows time slice capabilities, thus their execution time.
- Temporal Isolation – A process's execution time depends only on its capabilities.
- Low-overhead – Scheduling decided by a lookup table.
- Dynamic – Process can alter their time slices.

[2] Flush cache, branch predictors, etc., support is hardware dependant.

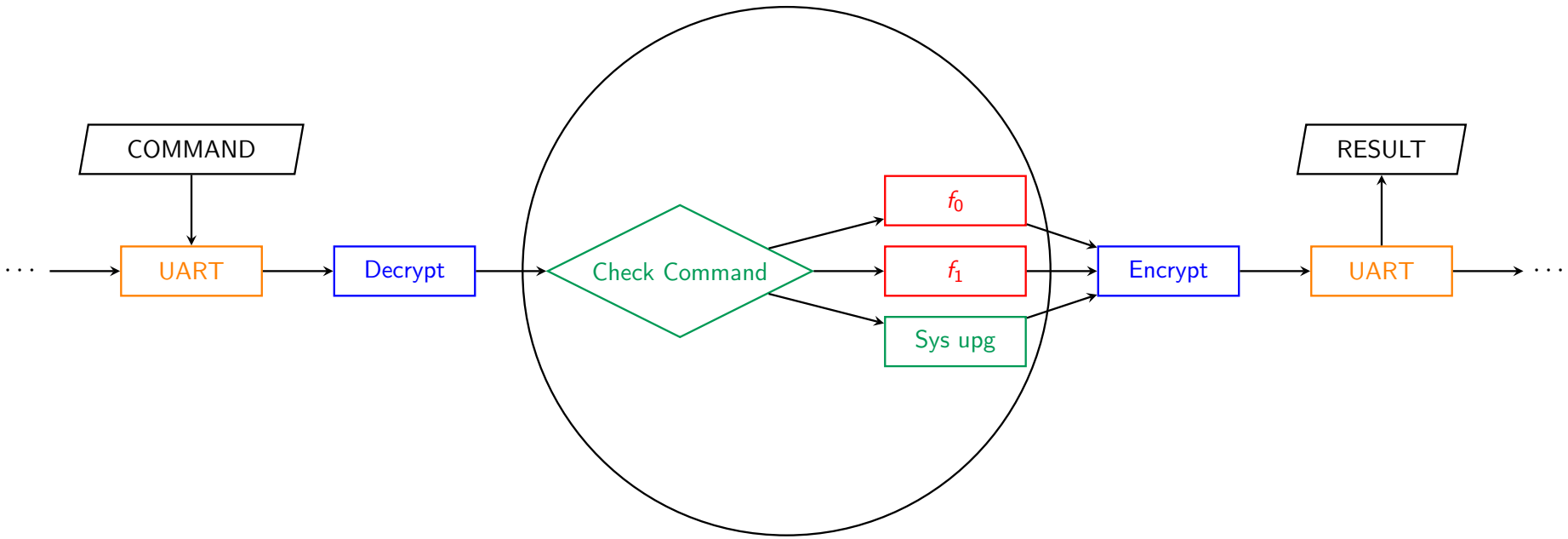
Time Slice Capability



- hartid – ID of a hardware thread.[3]
- begin – start of a time slice
- free – start of minor frame
- end – end of a time slice and minor frame

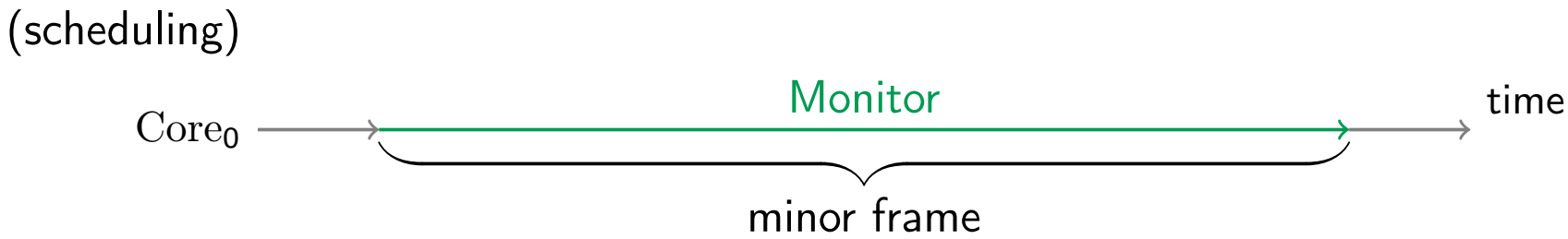
[3] Hardware Thread – Logically separate processor.

Proof-of-Concept with Time Slices



Proof-of-Concept with Time Slice

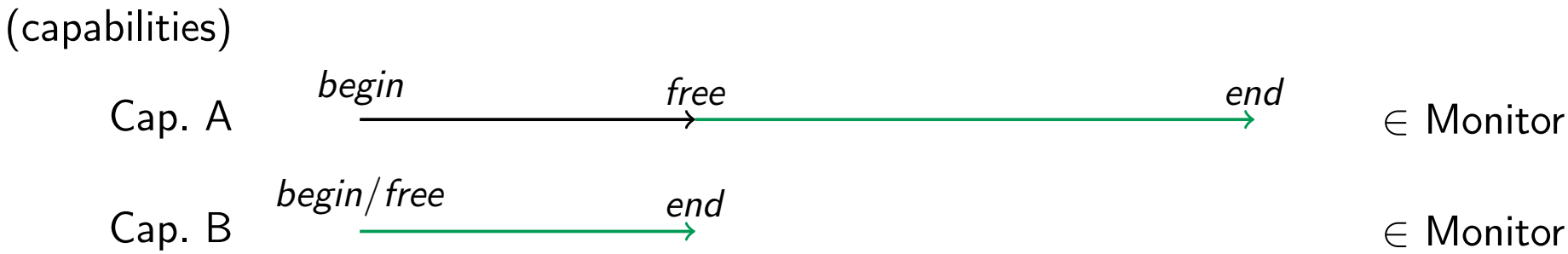
Monitor has the initial time slice

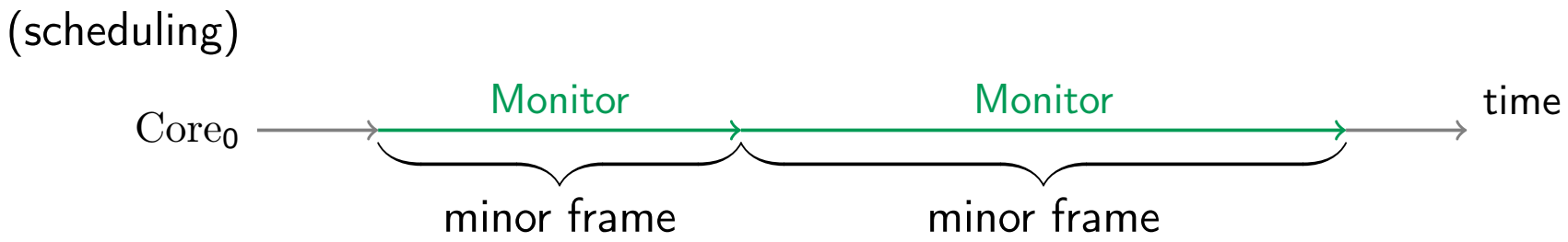


Proof-of-Concept with Time Slice

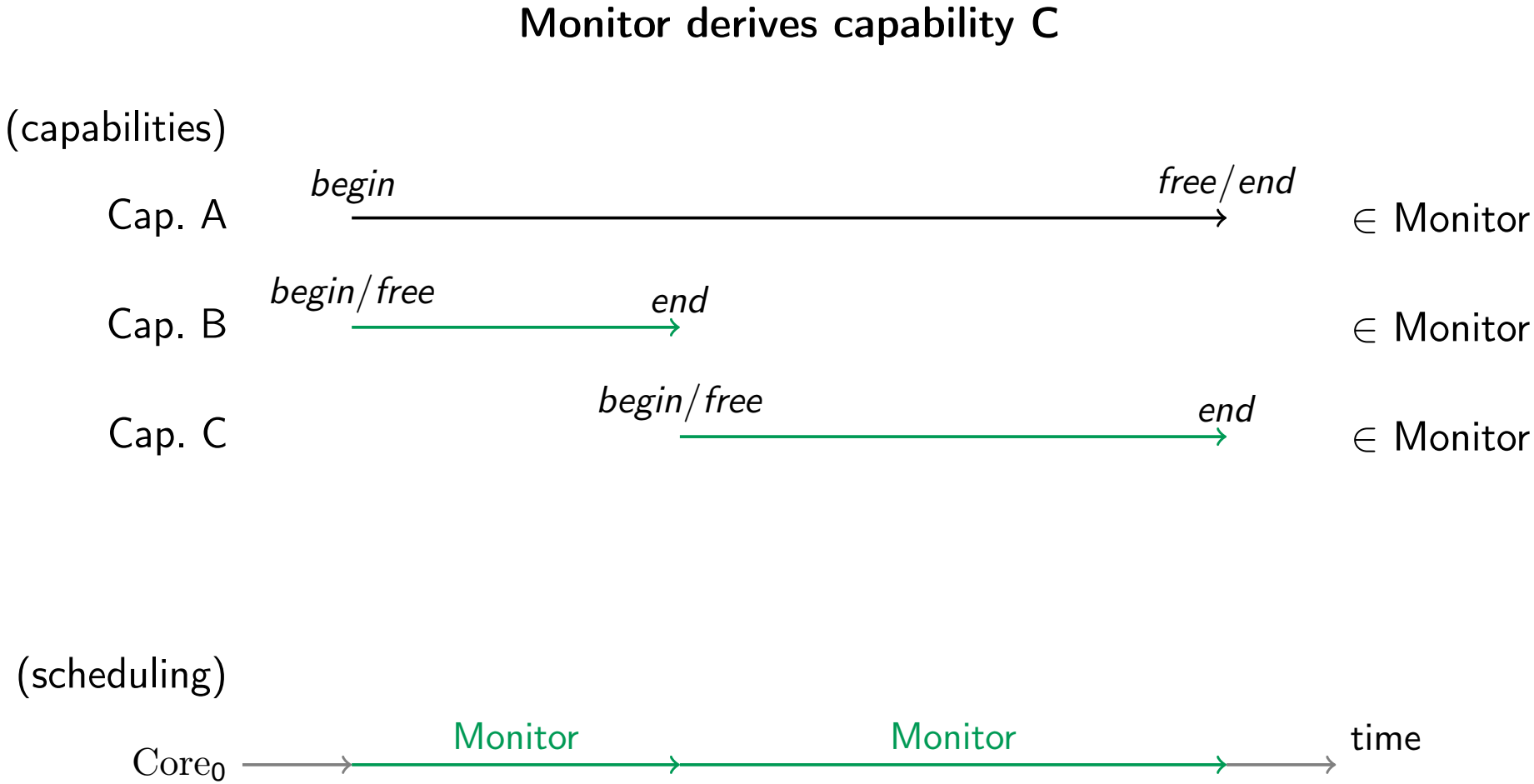
Monitor derives capability B

(only create slices from *free* to *end*)

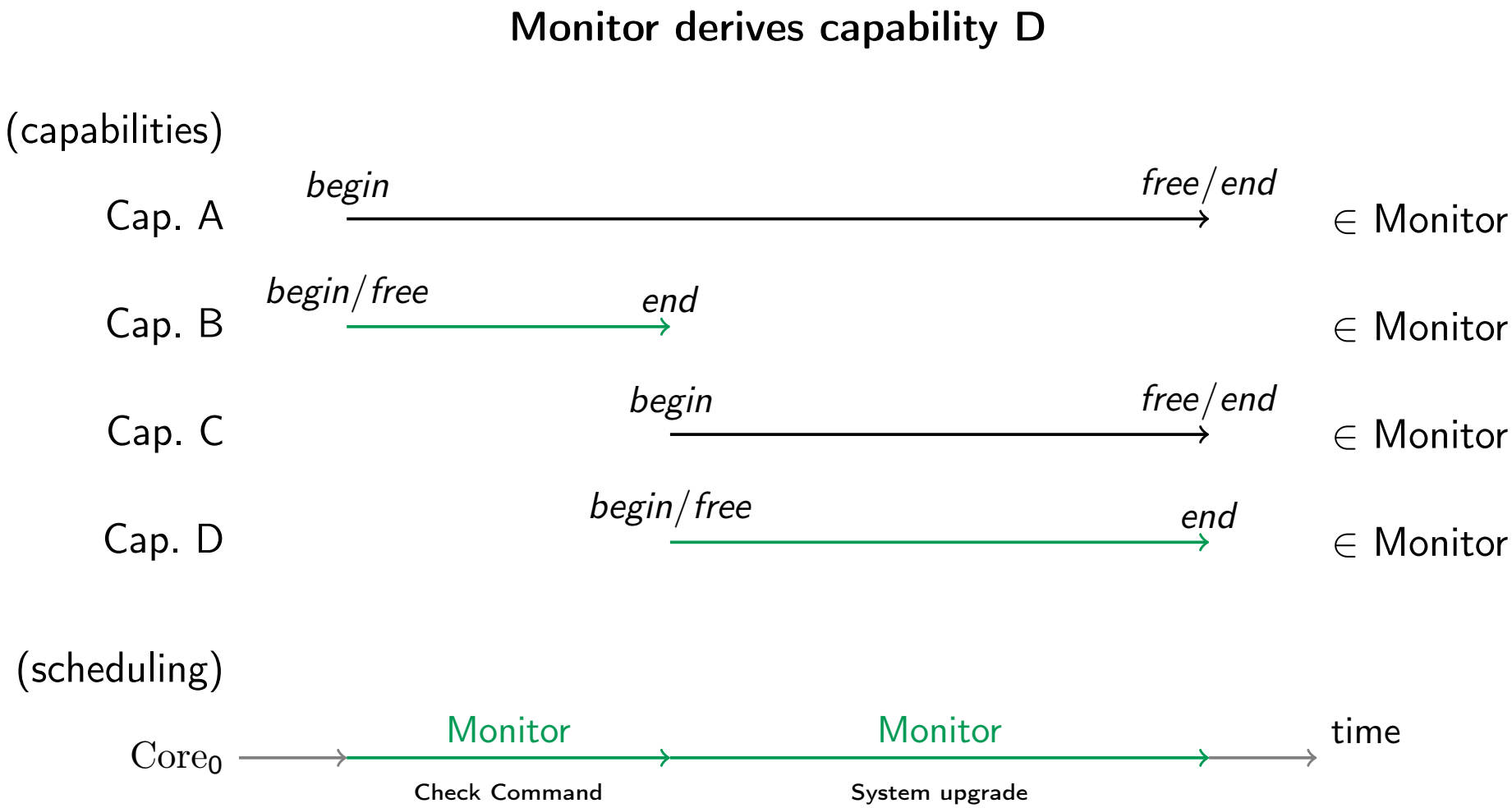




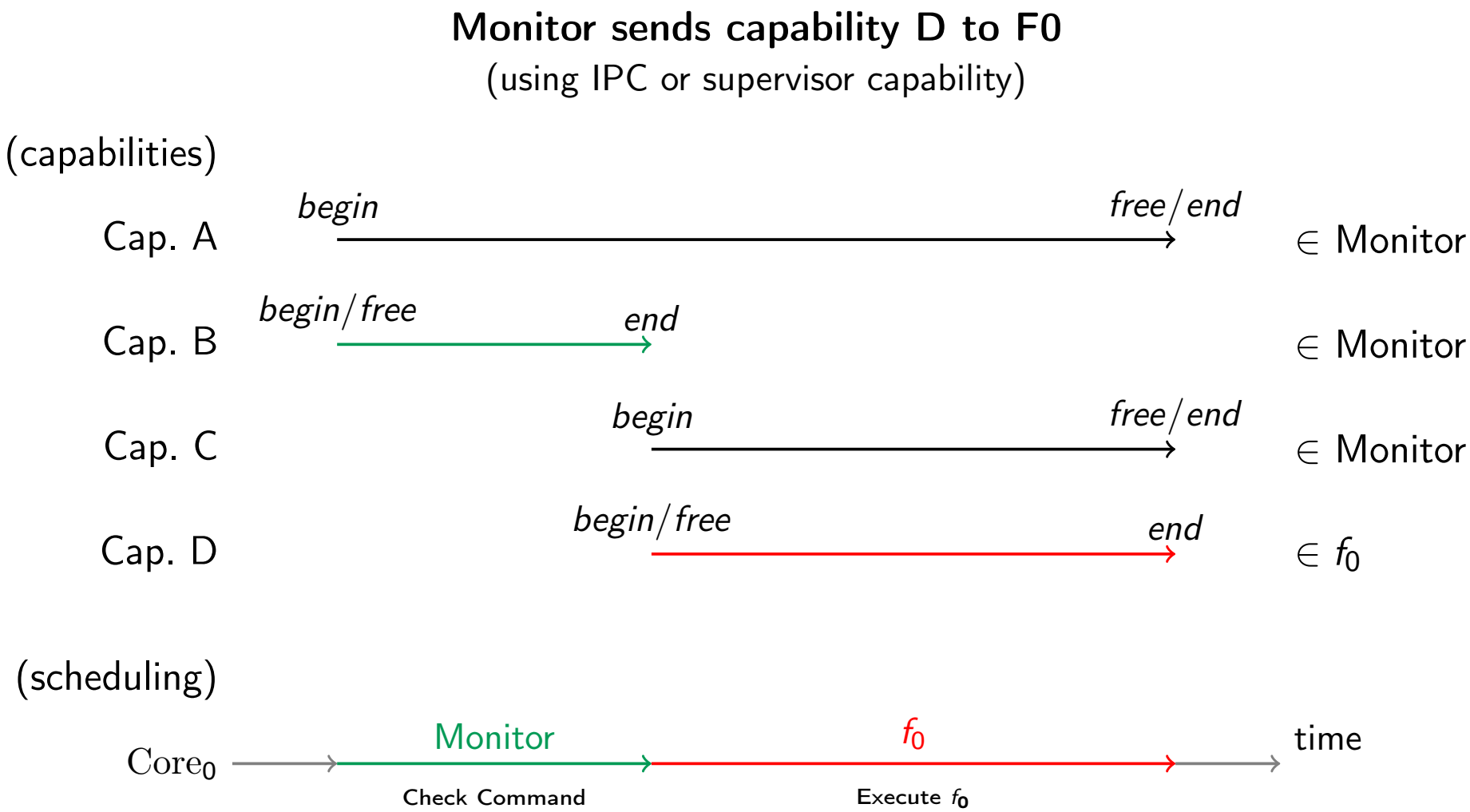
Proof-of-Concept with Time Slice



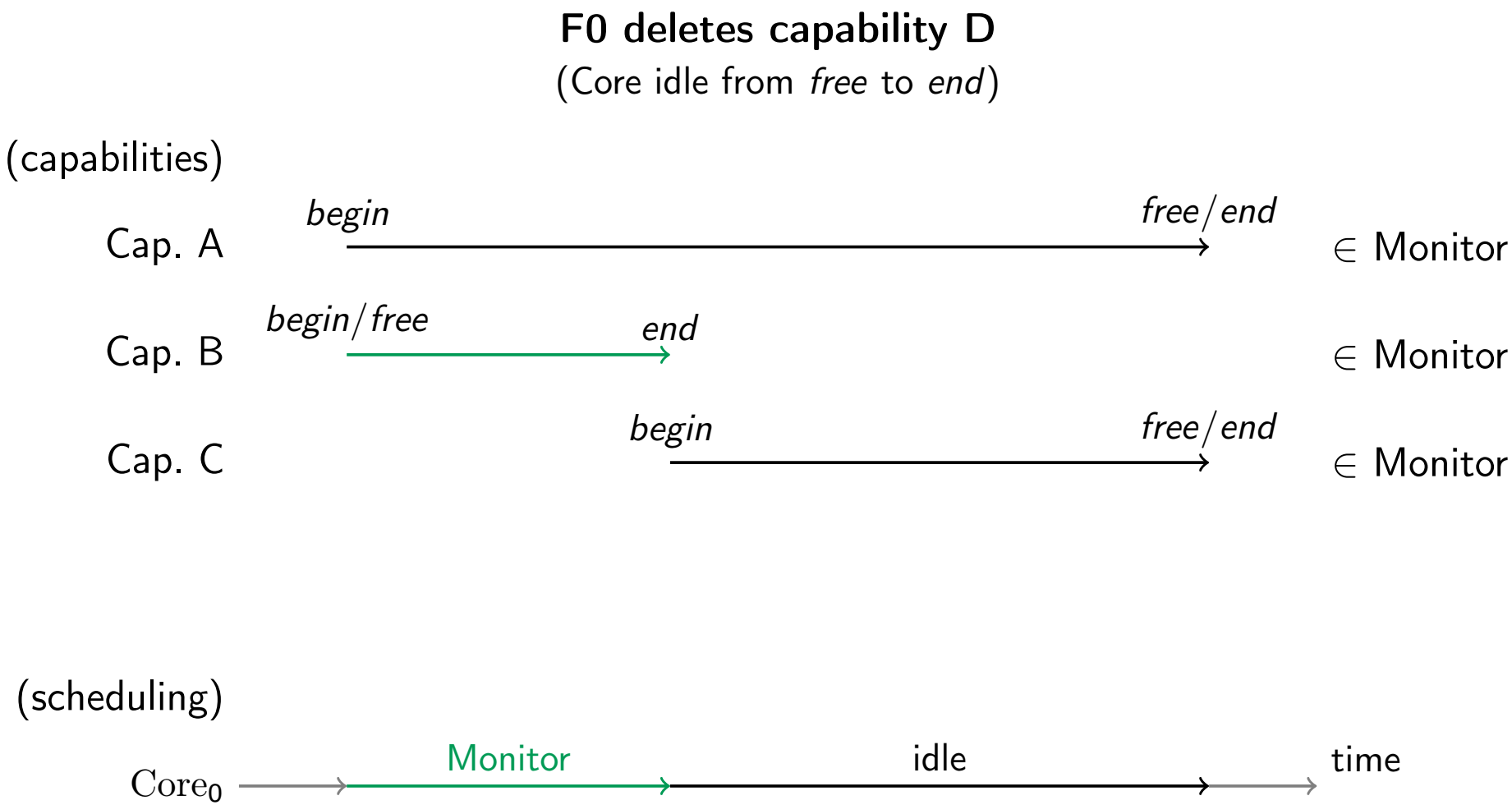
Proof-of-Concept with Time Slice



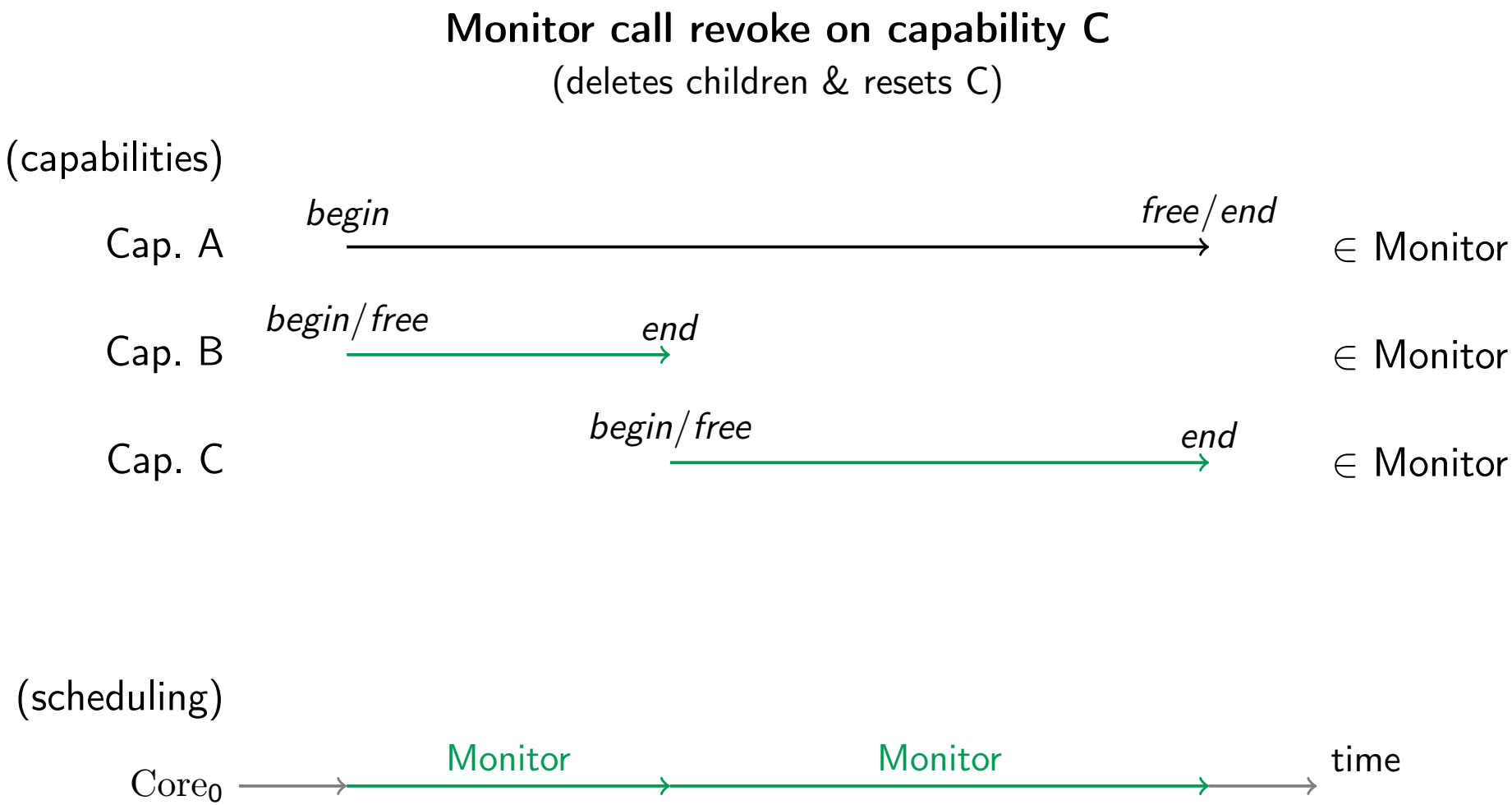
Proof-of-Concept with Time Slice



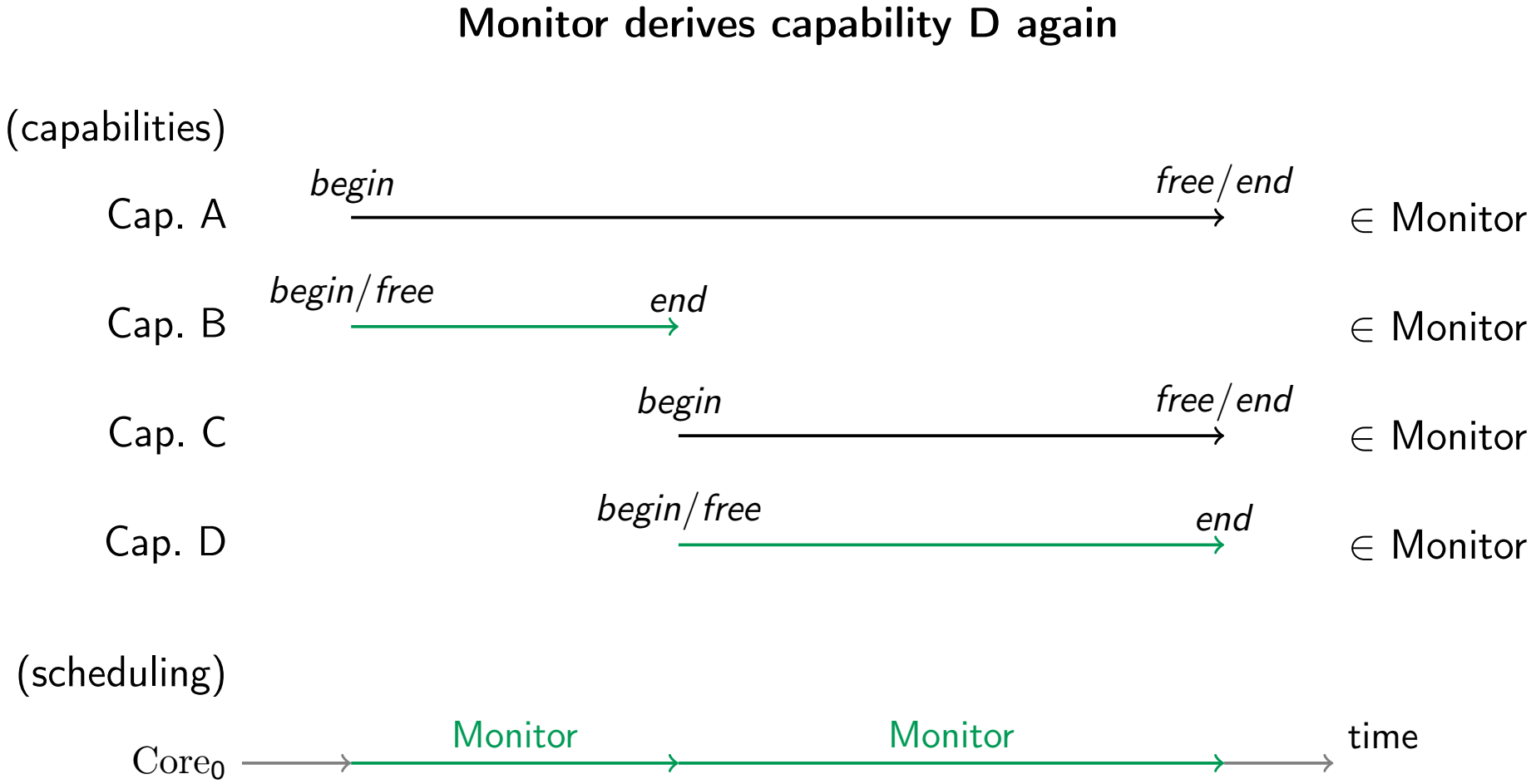
Proof-of-Concept with Time Slice



Proof-of-Concept with Time Slice

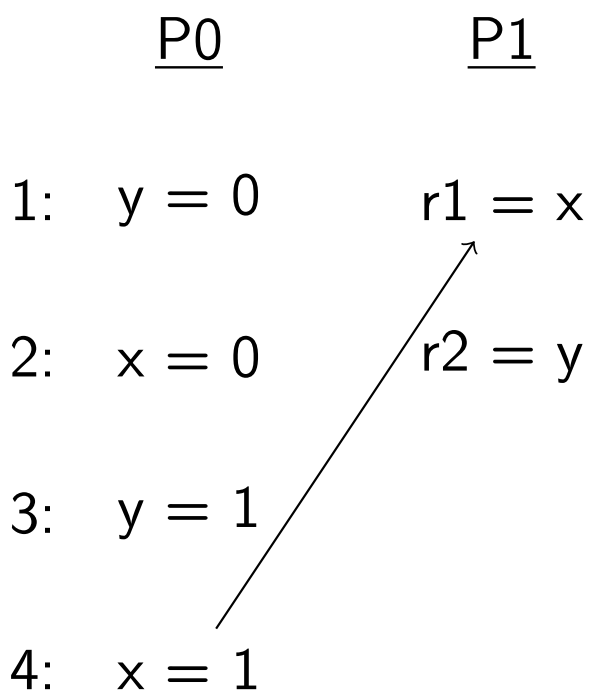


Proof-of-Concept with Time Slice

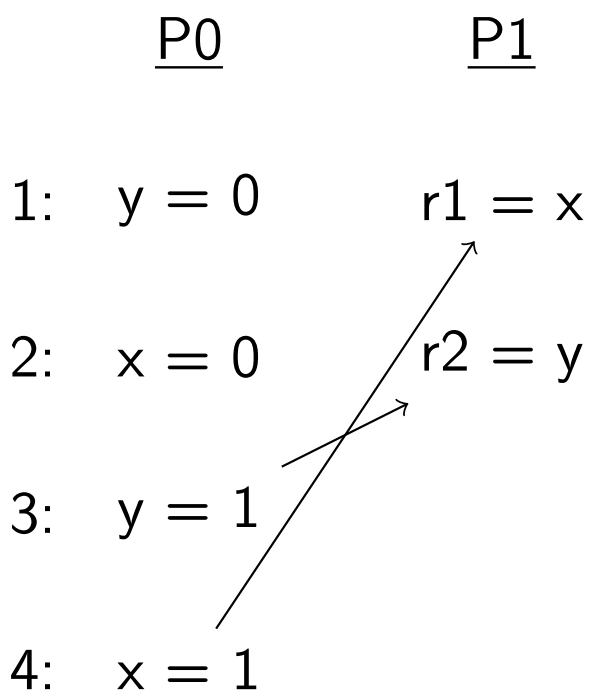


Multicore HolBA & Kernel Verification

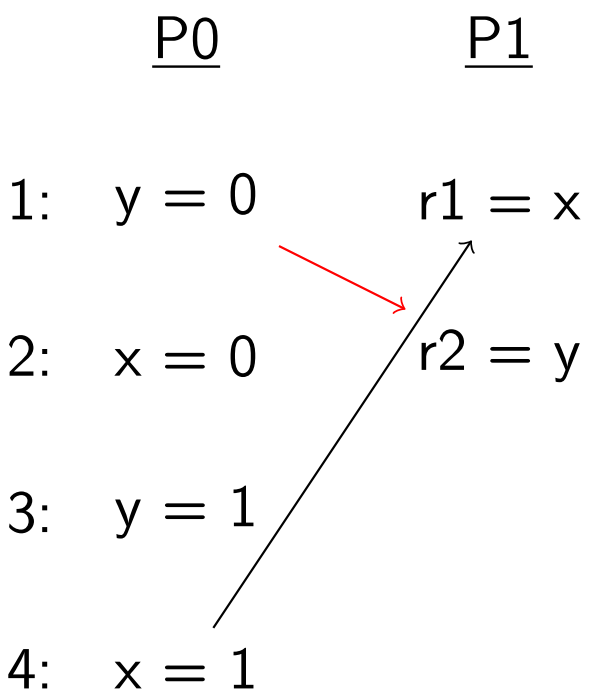
Multicore HolBA



Multicore HolBA

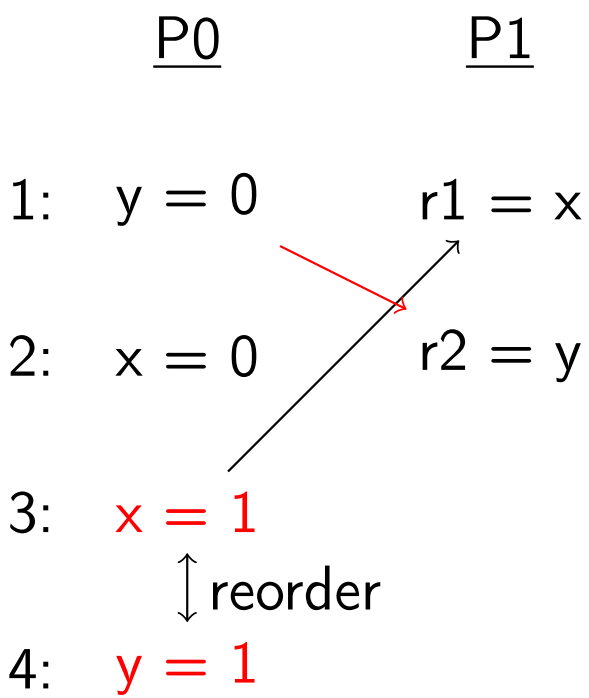


Multicore HolBA



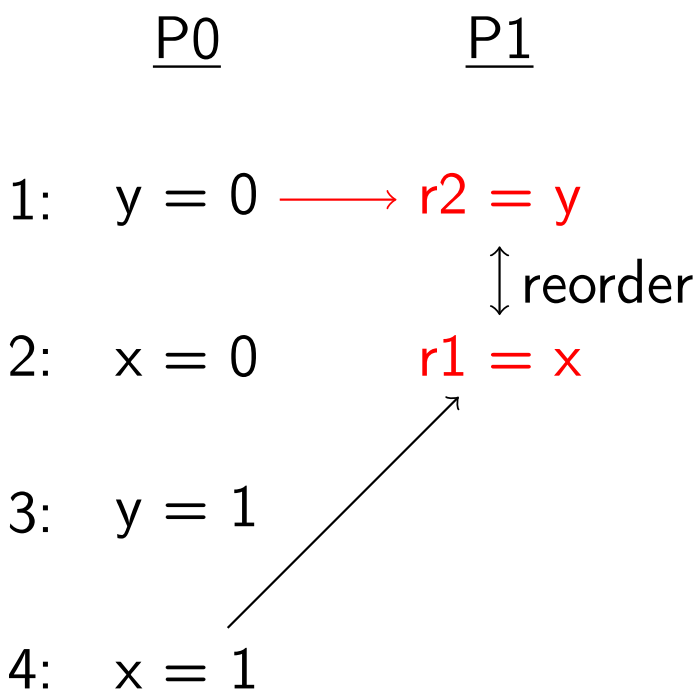
Memory operations may be reordered in RISC-V.

Multicore HolBA



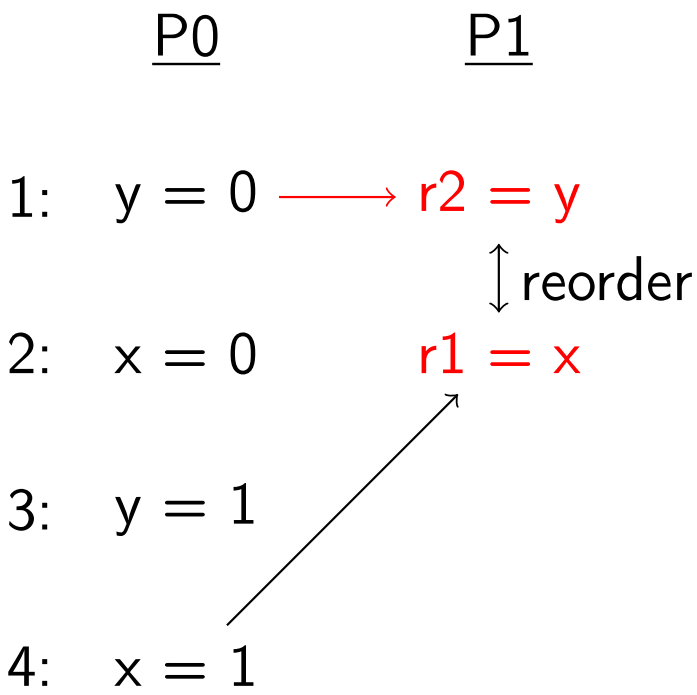
Writes may be reordered.

Multicore HolBA



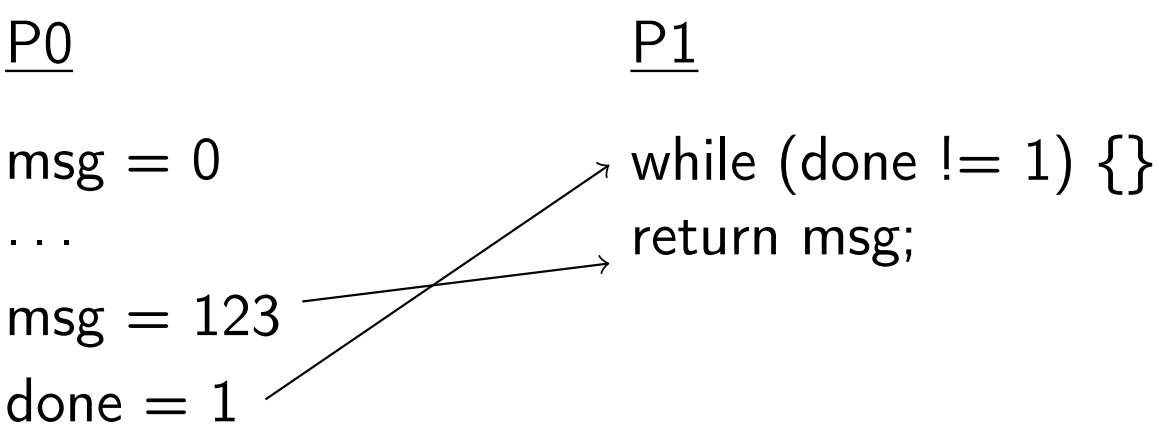
Reads may be reordered.

Multicore HolBA

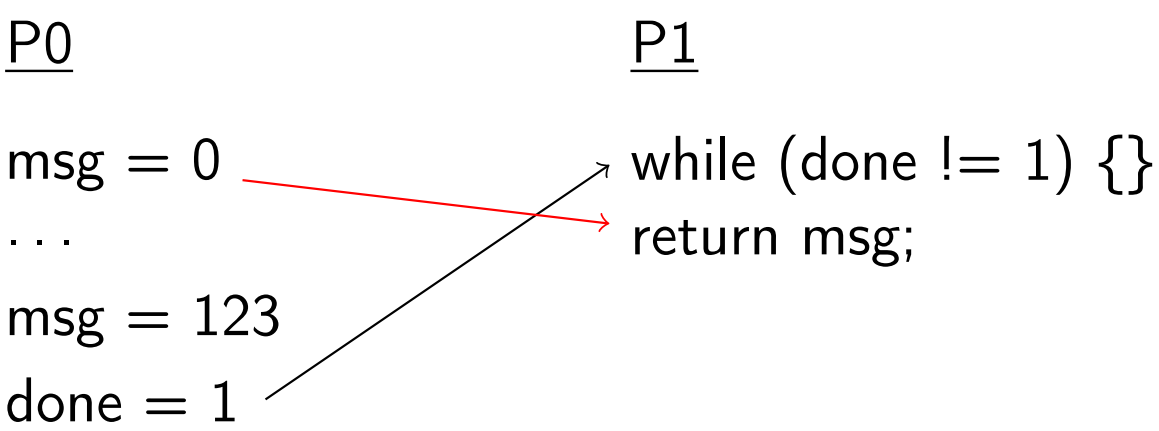


Verification of multicore RISC-V need all reorderings.
Multicore HolBA provides this!

Separation Kernel Example



Separation Kernel Example



Research Plan

Research Plan

- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept

[4] Worst-case execution time

Research Plan

- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept
- Spring 2023
  - Publication of multicore HolBA
  - Measurements of WCET[4] and jitter of non-preemptive kernel parts
  - Implement secure interrupts, optimized scheduler, and 32-bit kernel version

[4] Worst-case execution time

Research Plan

- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept
- Spring 2023
  - Publication of multicore HolBA
  - Measurements of WCET[4] and jitter of non-preemptive kernel parts
  - Implement secure interrupts, optimized scheduler, and 32-bit kernel version
- Summer 2023
  - Model and proofs on some concurrent kernel code using multicore HolBA
  - Finish sequential high-level model of kernel

[4] Worst-case execution time

Research Plan

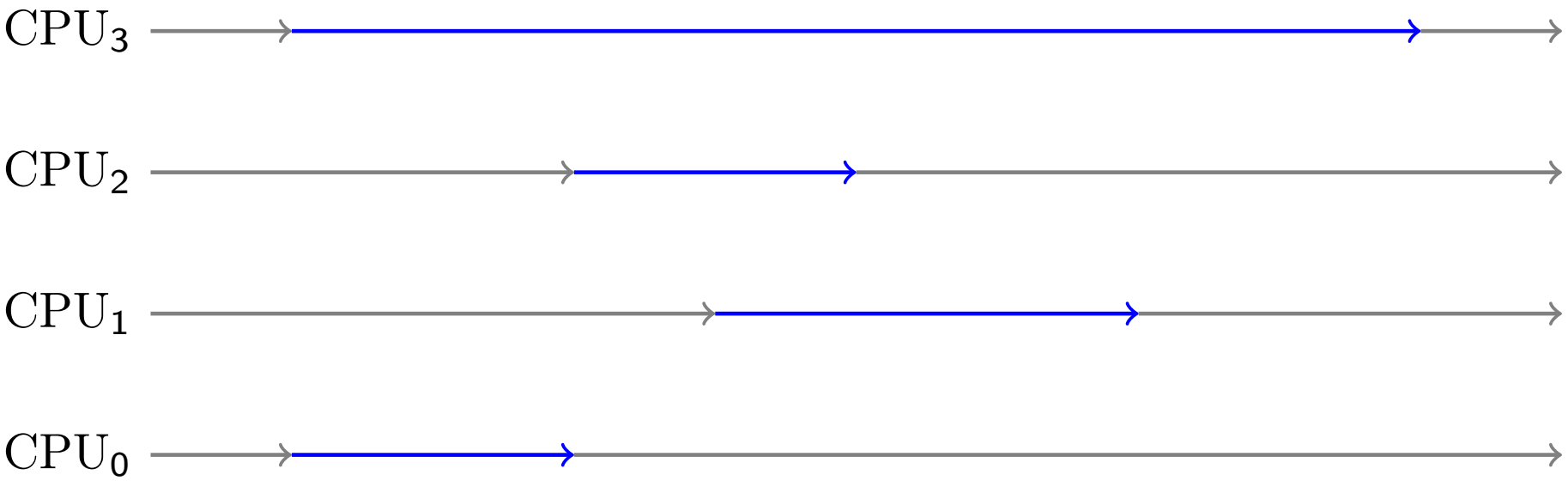
- Dec. 2022 – Complete proof-of-concept
- Jan./Feb. 2023 – Evaluation and publication of kernel with proof-of-concept
- Spring 2023
  - Publication of multicore HolBA
  - Measurements of WCET[4] and jitter of non-preemptive kernel parts
  - Implement secure interrupts, optimized scheduler, and 32-bit kernel version
- Summer 2023
  - Model and proofs on some concurrent kernel code using multicore HolBA
  - Finish sequential high-level model of kernel
- Autumn 2023 – Finish concurrent high-level model of kernel

[4] Worst-case execution time

Questions?

Multicore Scheduling

Where does the blue process execute?



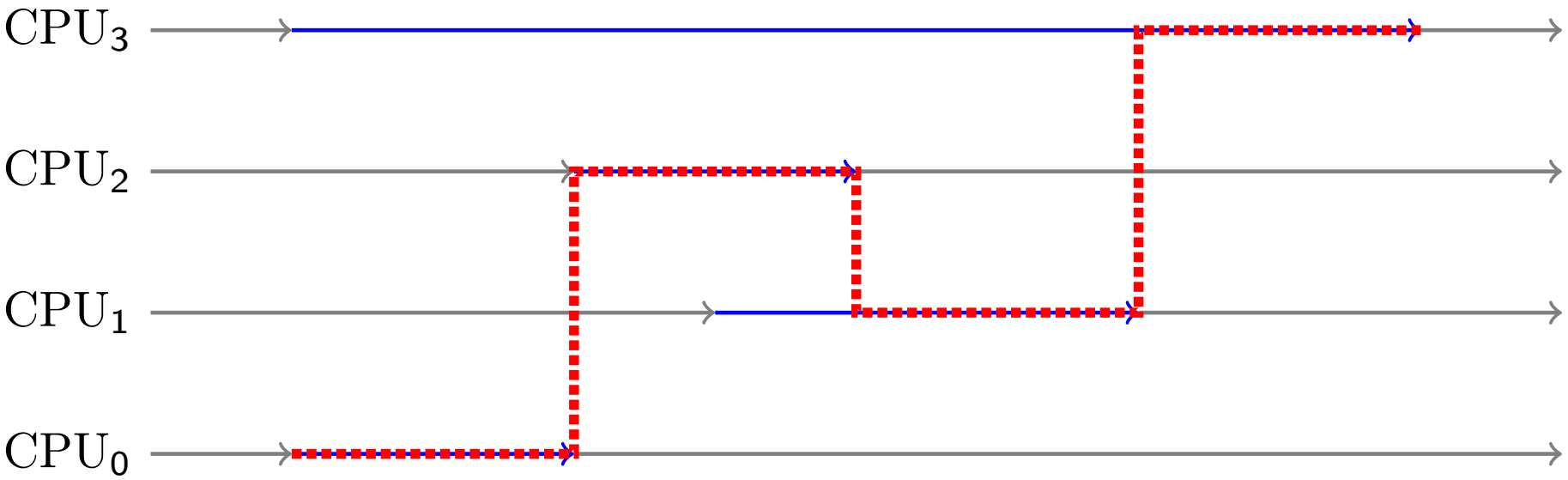
Process runs on one core, for determinism,[5] we need priority rules.

- currently running core
- core with smallest ID.

[5] non-determinism may leak information

Multicore Scheduling

Where does the blue process execute?



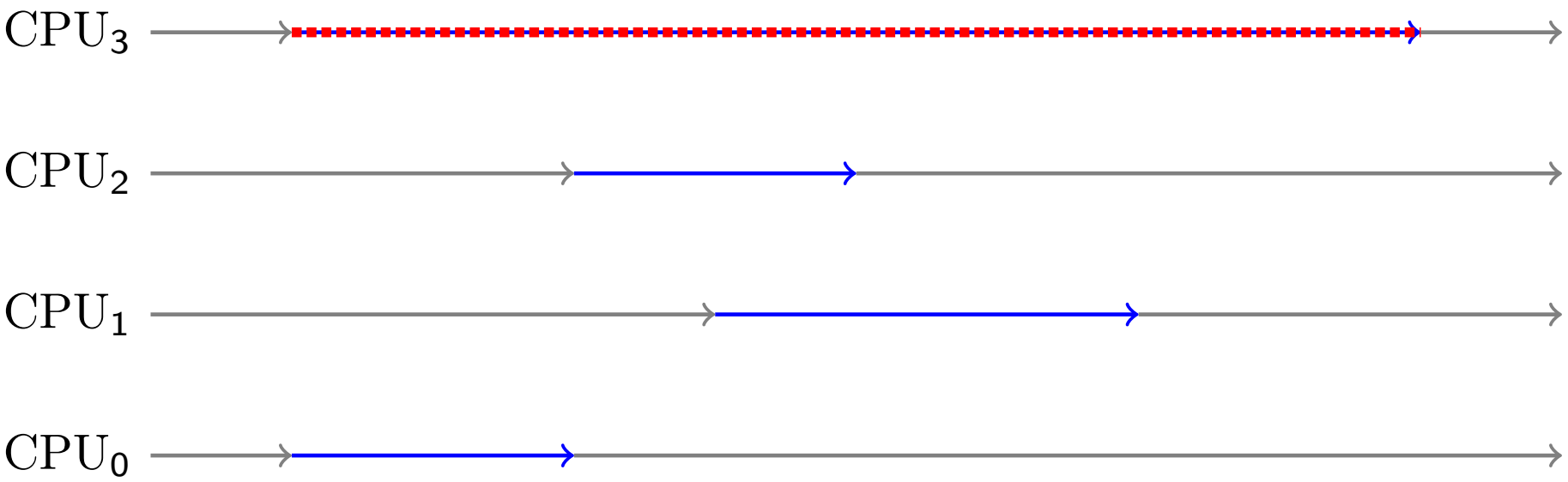
Process runs on one core, for determinism,[5] we need priority rules.

- currently running core
- core with smallest ID.

[5] non-determinism may leak information

Multicore Scheduling

Where does the blue process execute?



Process runs on one core, for determinism,[5] we need priority rules.

- currently running core
- longest quantum (not implemented)
- core with smallest ID

[5] non-determinism may leak information