CHERIoT

Robert Norton-Wright
Saar Amar, Tony Chen, David Chisnall,
Wes Filardo, Kunyan Liu, Hongyan Xia
Microsoft

IoT

The ‘S’ stands for security

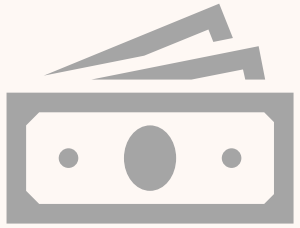
Motivation – IoT and embedded

The IoT ecosystem:

Includes diverse codebases

Mostly unsafe C/C++

Mitigations are rare



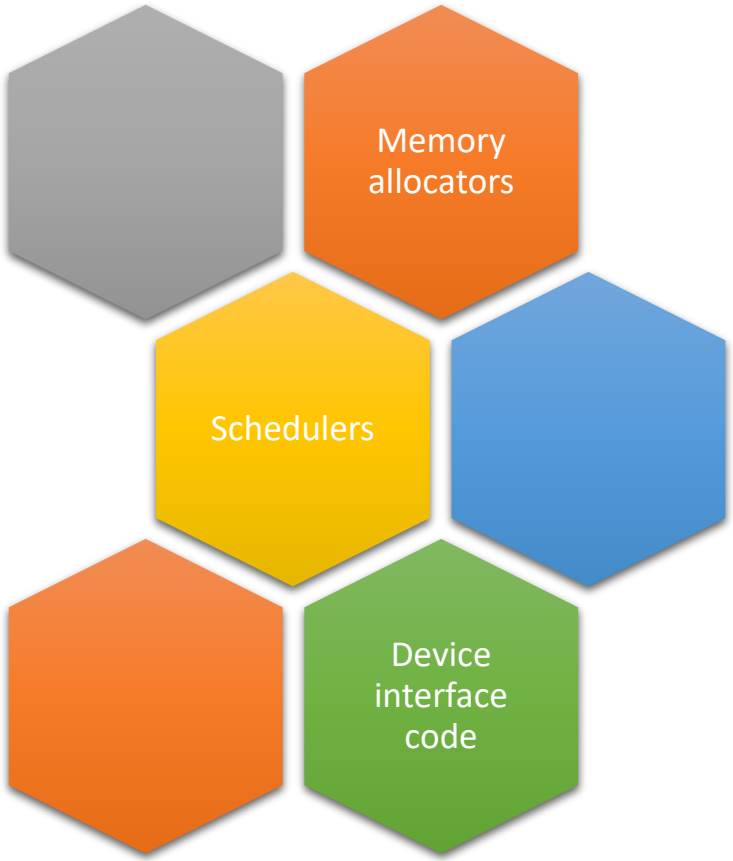
Rewriting in safe languages has challenges:

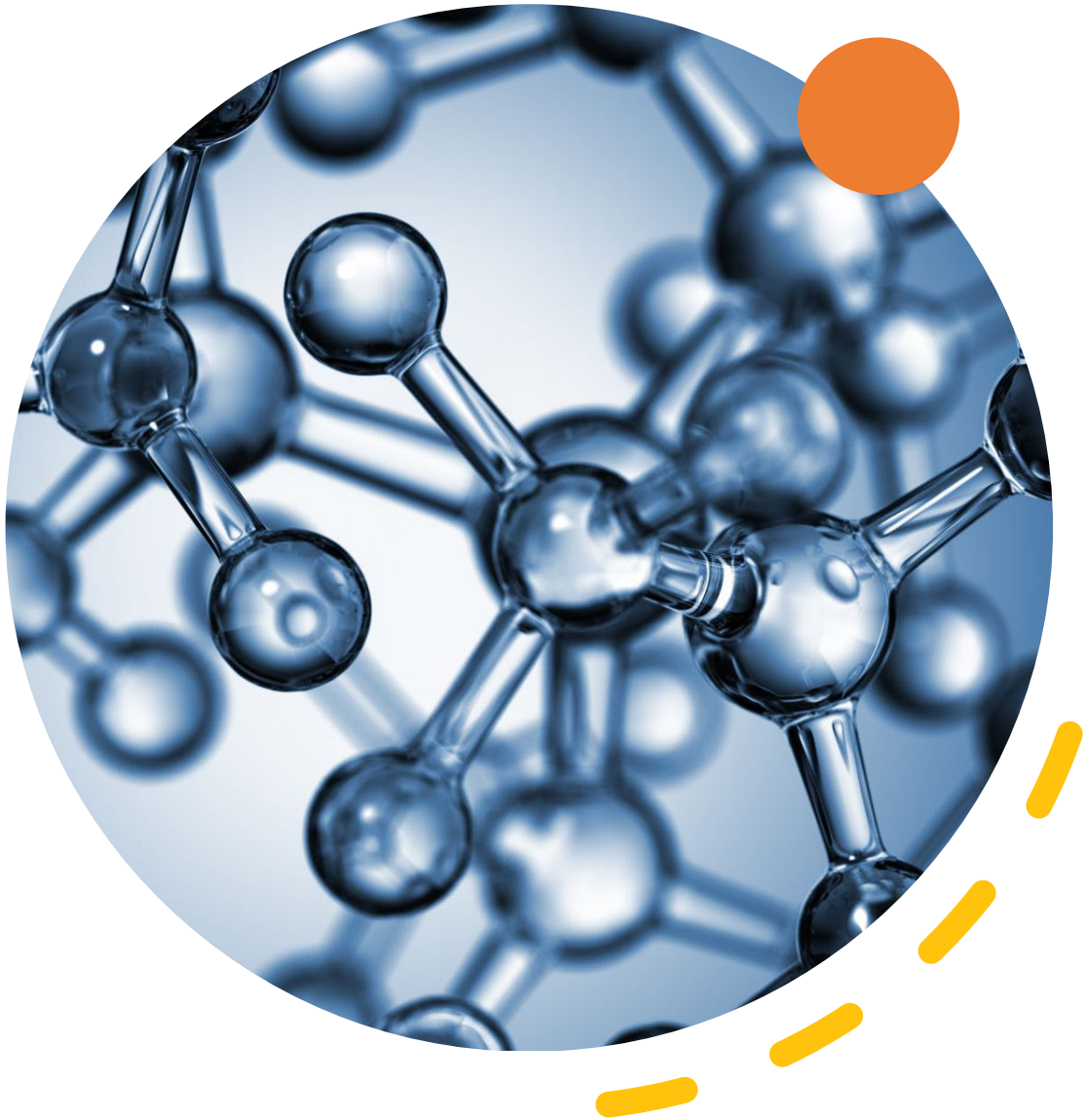
Expensive

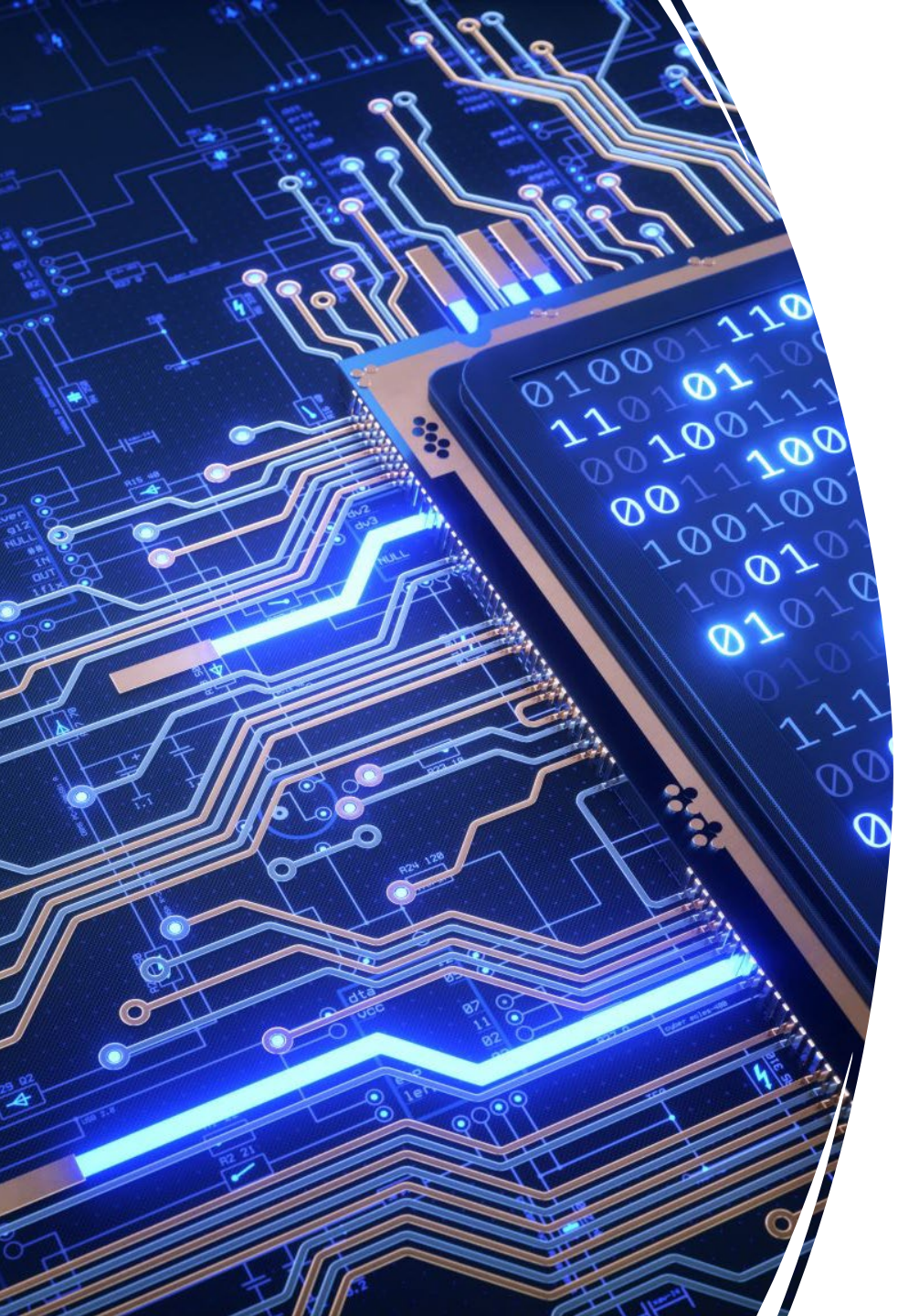
Talent shortage

Risk of introducing bugs

Much embedded code is intrinsically unsafe







Some things work to our advantage

- Full control of software
  - Break compatibility, drop hybrid mode, simplify ISA
- Very fast tightly coupled memory
  - Enables new temporal safety mechanism

CHERIoT shrinks metadata to 32 bits

Bounds	• No guaranteed out-of-bounds range
Sealing	• Only 3 bits of sealing type • Separate code and data sealing spaces
Permissions	• 12 permissions in 6 bits

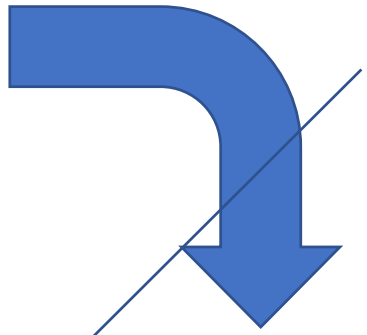
And we add things

Transitive permissions	• Permit-load-mutable, deep immutability • Permit-load-global, deep no-capture
Interrupt control via sentries	• Jumping to these enables / disables interrupts
Temporal safety via a hardware revocation bitmap	• 1 bit per 8 bytes in a separate SRAM bank

Hardware load barrier adds temporal safety

- Load pointer computes the base address
- Looks up the corresponding revocation bit
- Invalidates the pointer if the memory is freed

```
void *x = malloc(42);
// Print the allocated value:
Debug::log("Allocated: {}", x);
free(x);
// Print the dangling pointer
Debug::log("Use after free: {}", x);
```



Valid bit cleared, *any* attempt to use as a pointer will trap

```
Allocating compartment: Allocated: 0x80005900 (v:1 0x80005900-0x80005930 l:0x30 o:0x0 p: G RWcgm- -- ---)
Allocating compartment: Use after free: 0x80005900 (v:0 0x80005900-0x80005930 l:0x30 o:0x0 p: G RWcgm- -- ---)
```

Baseline security guarantees

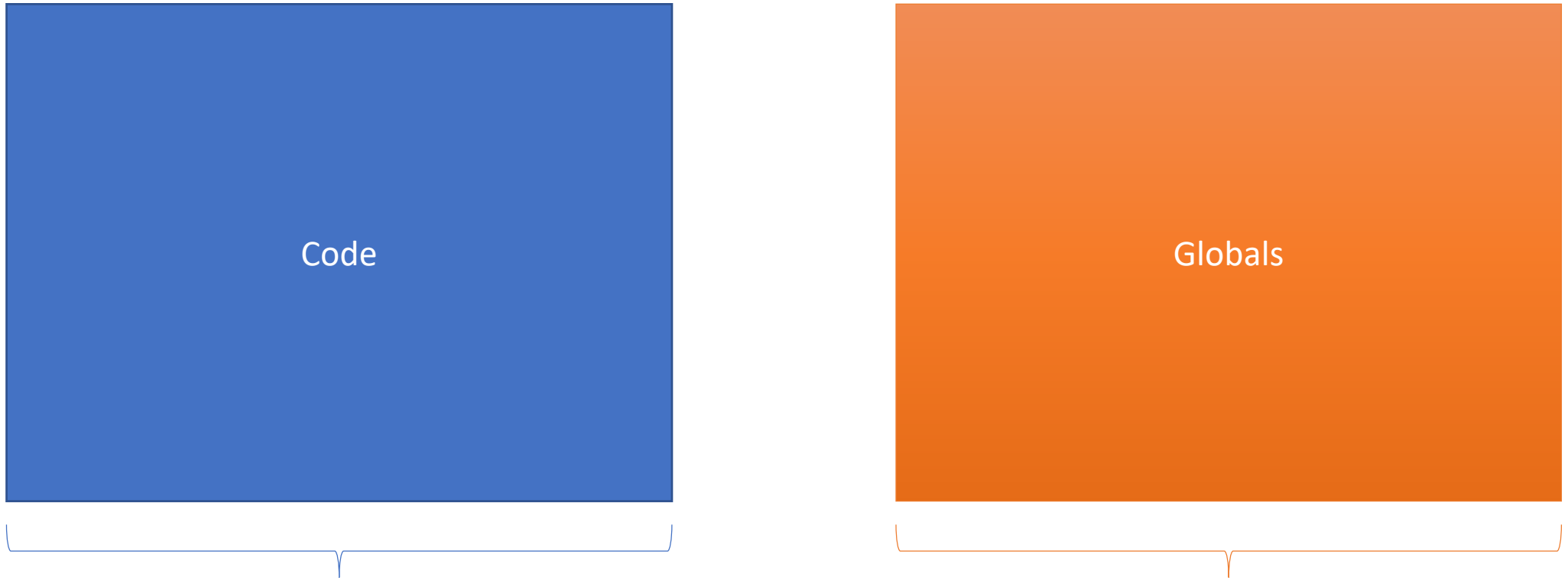
No pointer injection

No bounds violations

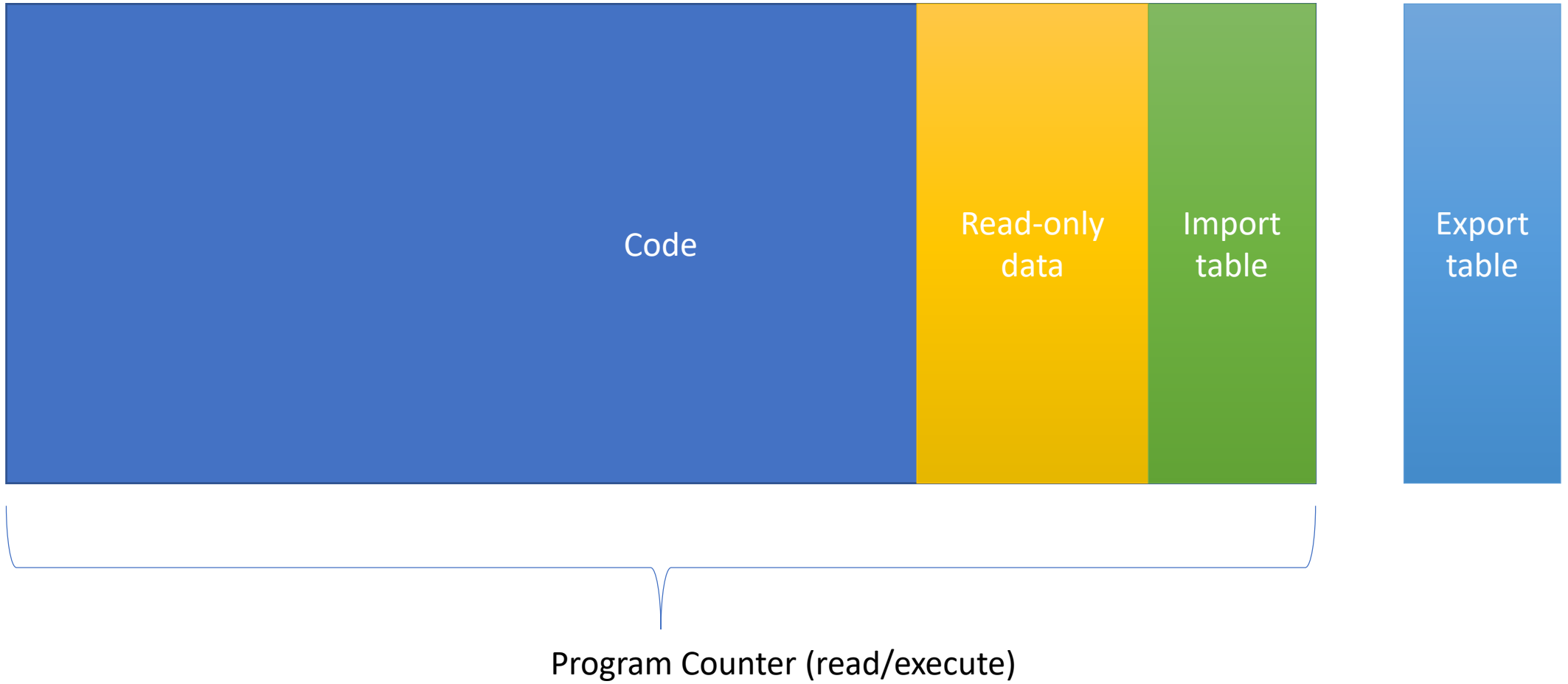
No use after free

The system can assume these for building higher-level abstractions.

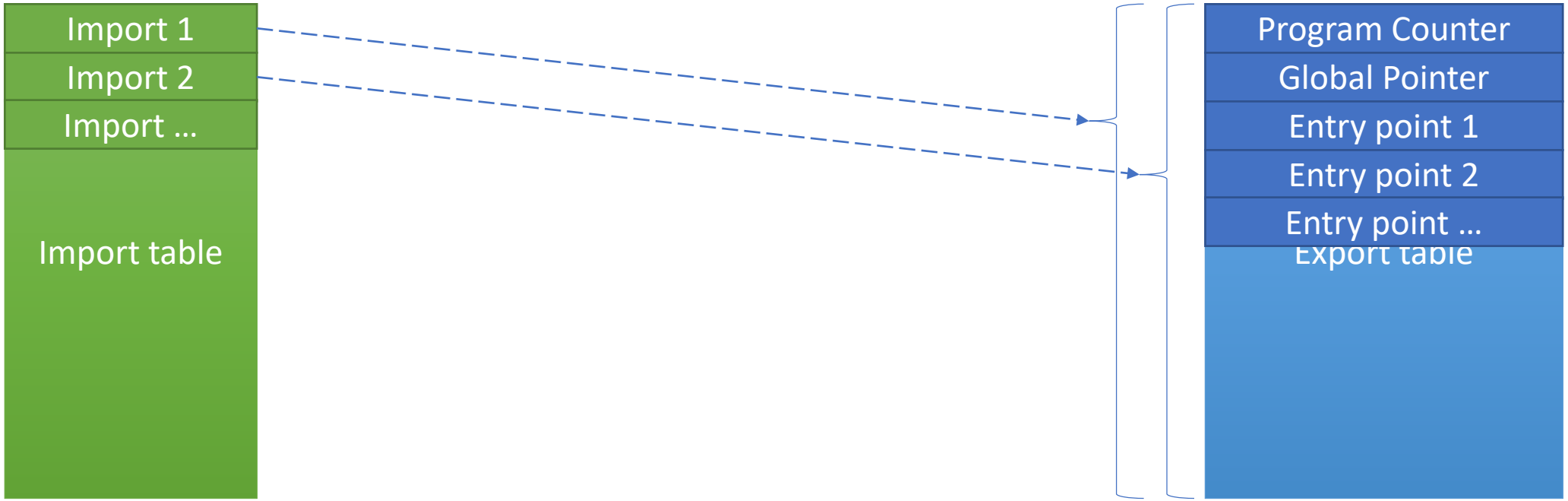
Compartments are code and data



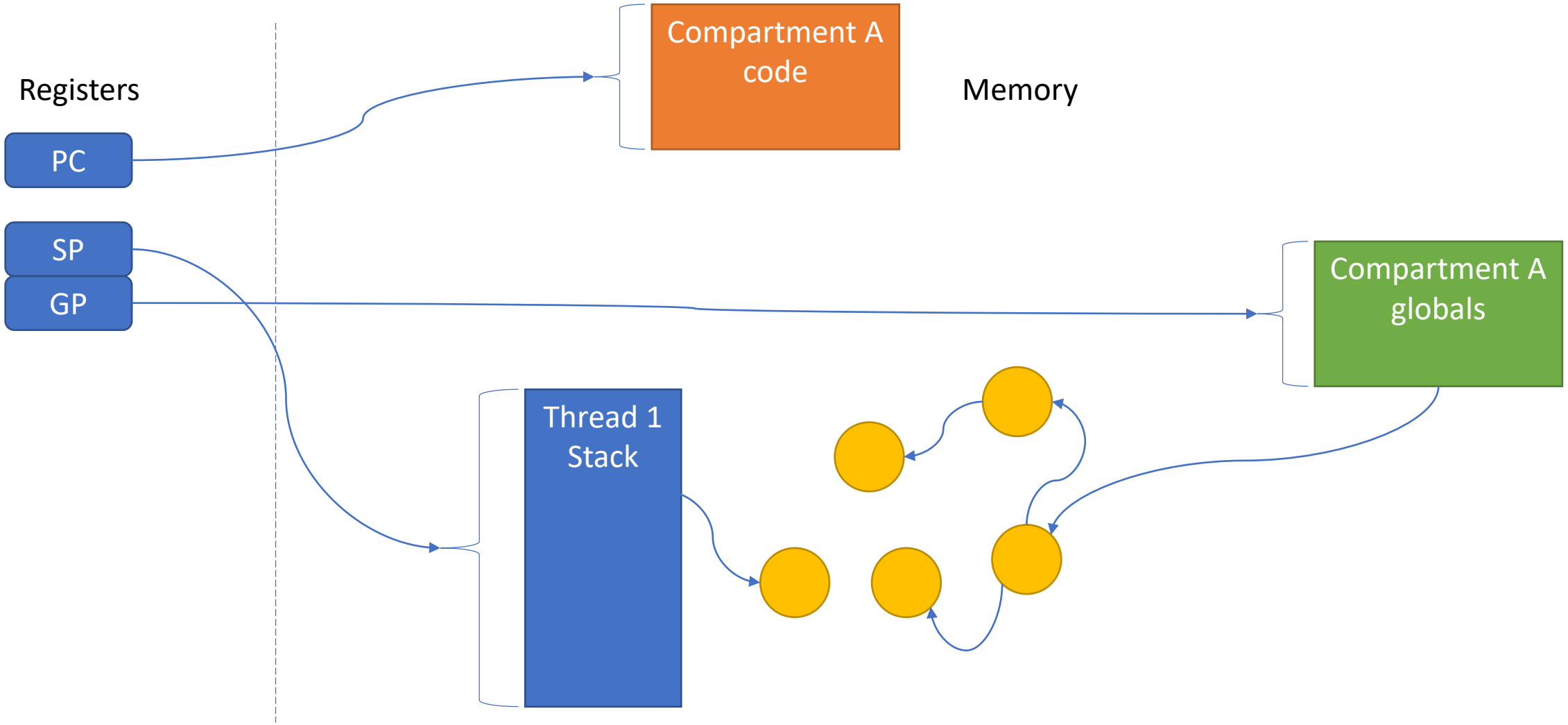
Compartments are code and data and exports



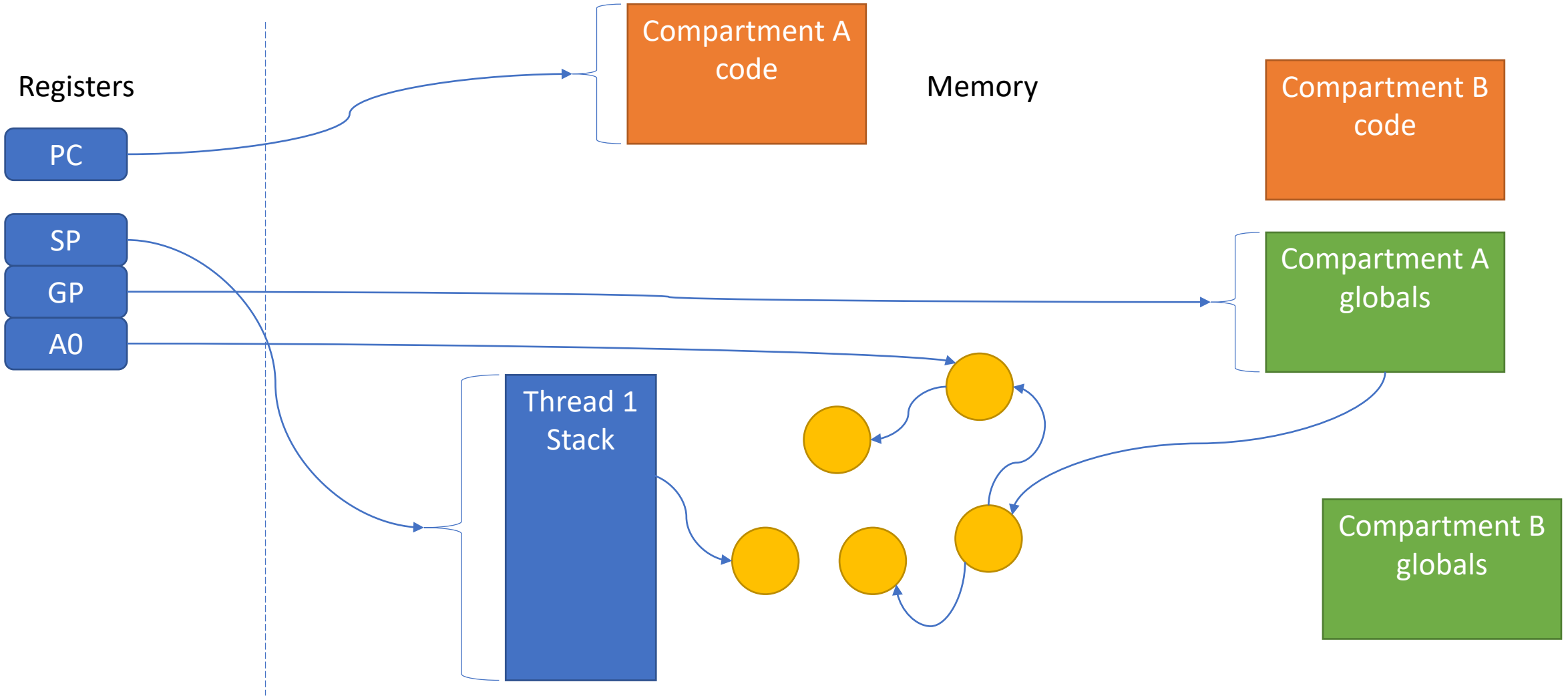
Compartments are code and data and exports



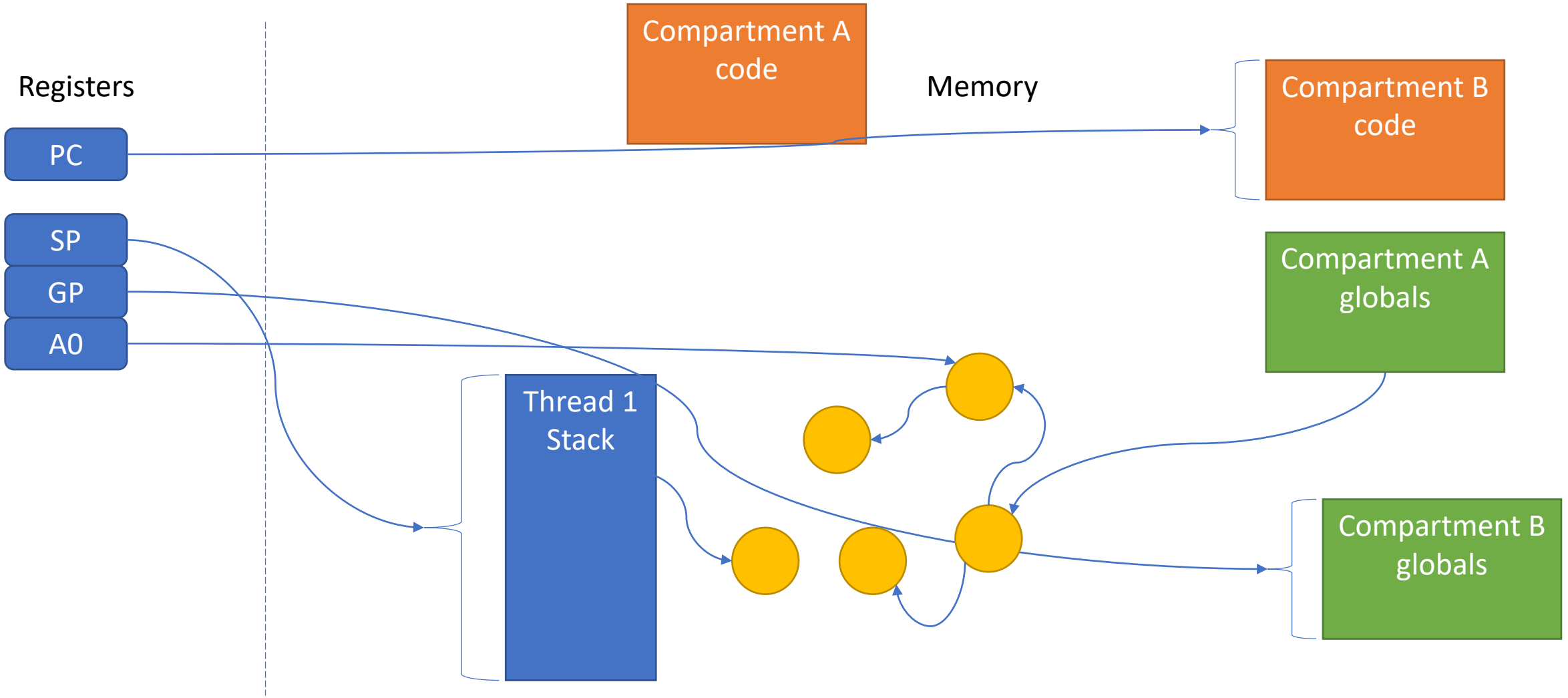
From unforgeable pointers to compartments



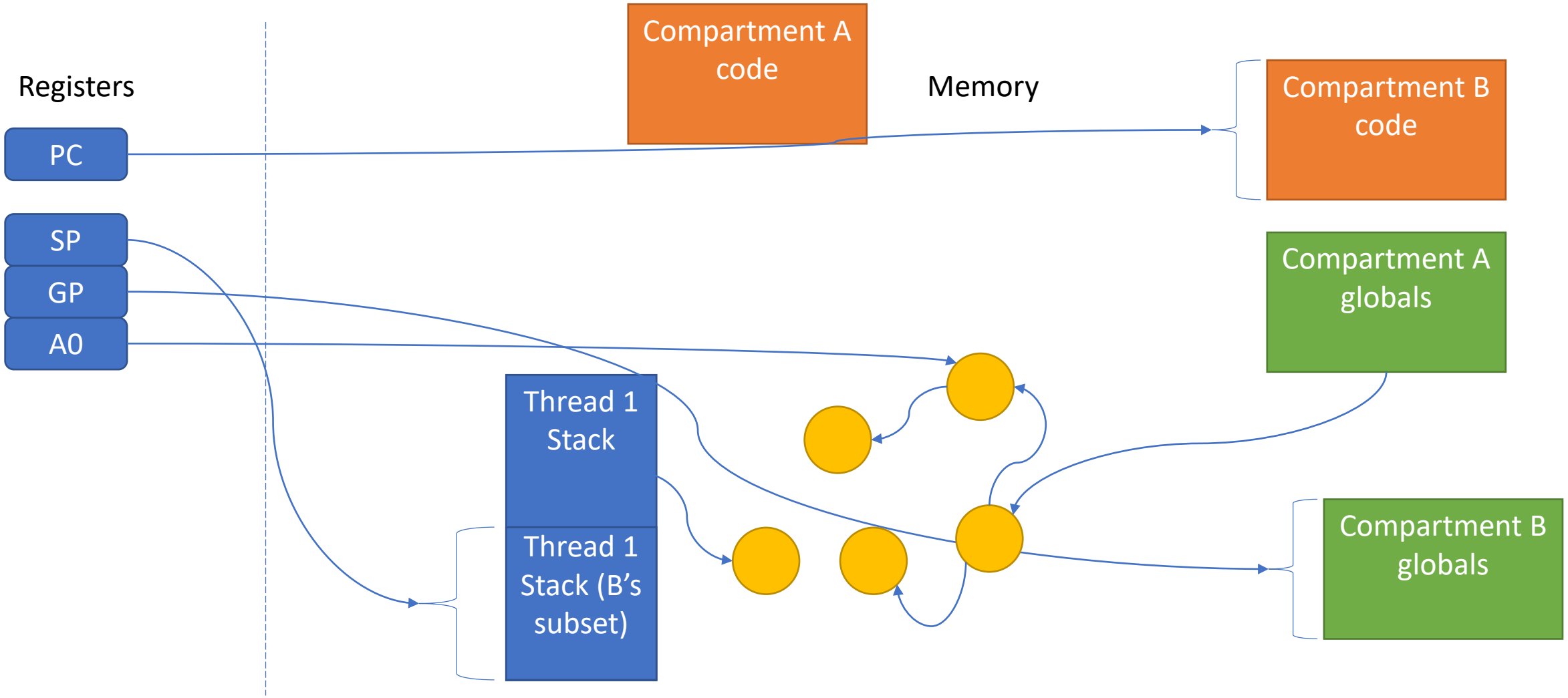
From unforgeable pointers to compartments



From unforgeable pointers to compartments



From unforgeable pointers to compartments



Security guarantees across compartments

No sharing except via explicit pointer passing

Pointers from the caller may prevent modification or capture

Trusted (privilege-separated) components

Loader

- Has full access to all memory
- Erases itself after boot
- Not needed if flash can store tags

Switcher

- Can see state from multiple threads and compartments
- Has access to a reserved register (and system registers)
- Around 300 instructions

Scheduler

- Trusted for availability
- No access to suspended thread state (registers or stack)

Memory allocator (optional)

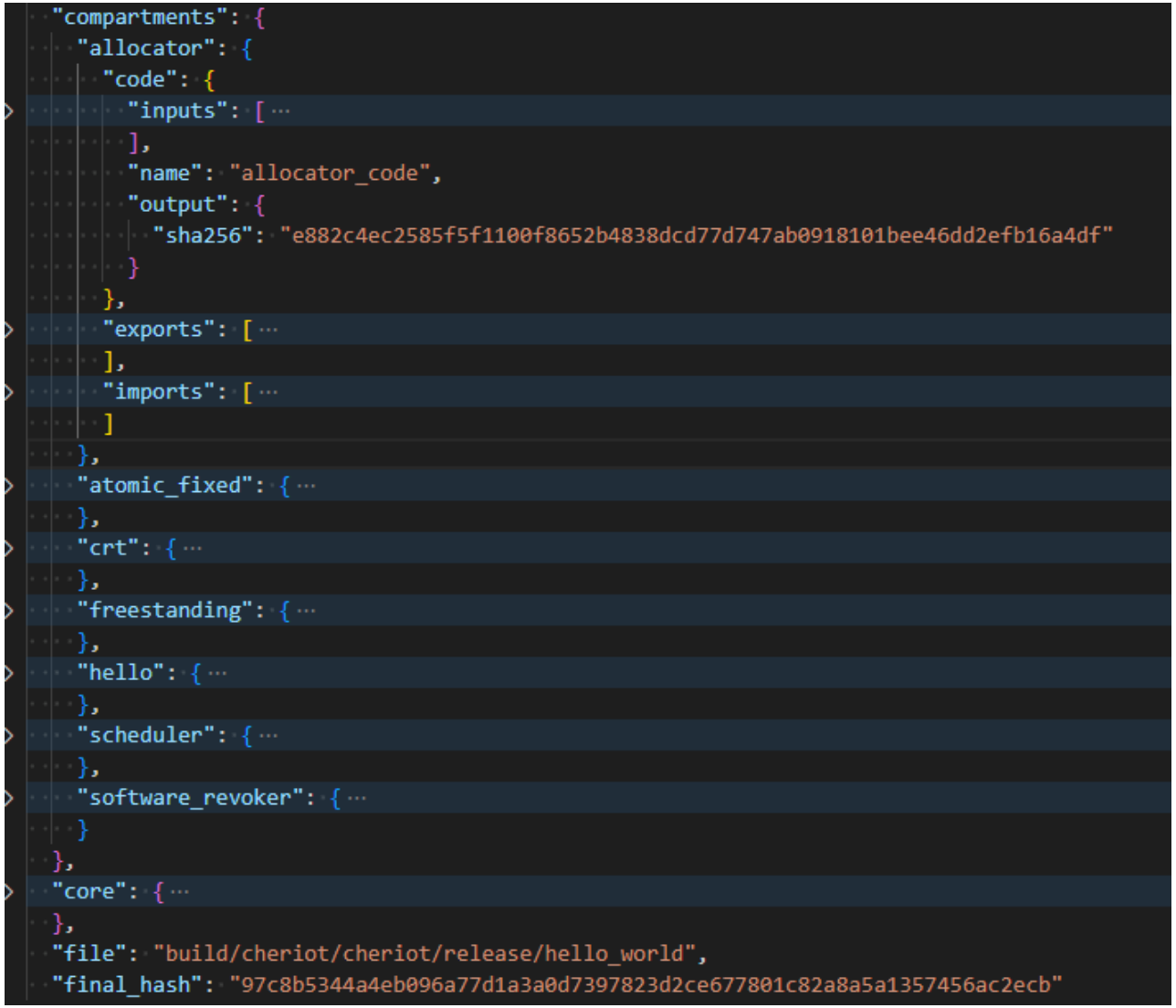
- Sets bounds / revocation state on allocations

Add compartmentalization to C/C++

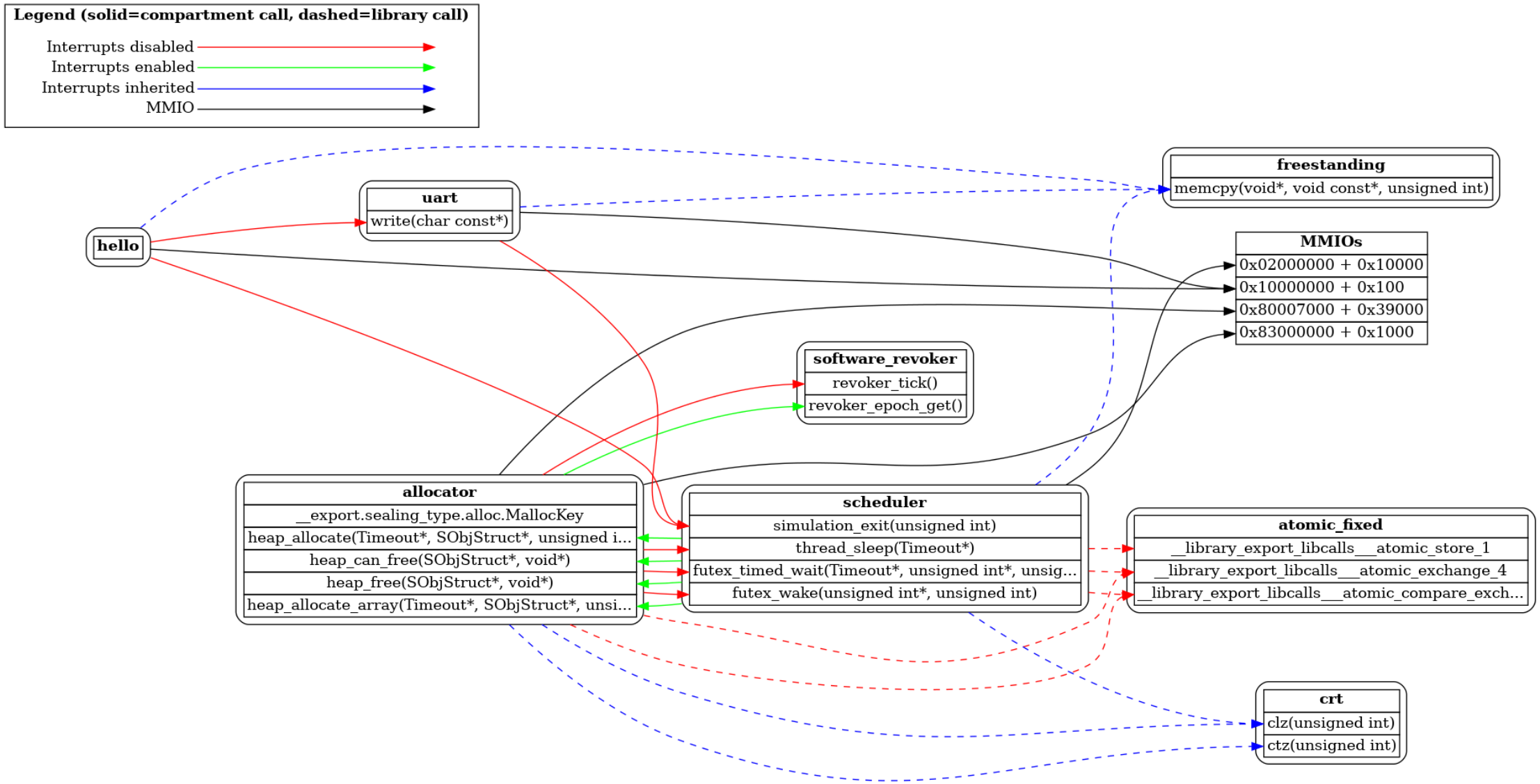
```
// Declaration adds an attribute to indicate
// the compartment containing the implementation
void __attribute__((cheri_compartment(“kv_store_sdk")))
publish(char *key, uint8_t *buffer, size_t size);

// Call site looks like normal C.
// Compiled to a direct call in compartments build with
// -cheri-compartment=kv_store_sdk
// Compiled to a cross-domain call in all other cases.
uint8_t buffer[BUFFER_SIZE];
publish("key_id", buffer, sizeof(buffer));
```

Linker reports



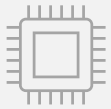
What can we statically audit?



Everything in this talk is open source

https://aka.ms/cheriot-tech-report

The ISA specification:
https://github.com/microsoft/cheriot-sail



The reference core:
https://github.com/microsoft/cheriot-ibex

The embedded OS:
https://github.com/microsoft/cheriot-rtos

The compiler (cheriot branch):
https://github.com/CTSRD-CHERI/llvm-project/

Thanks

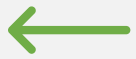
- UKRI / DSbD / CHERITech
- All prior CHERI work we've built on / inspired us:
  - CHERI-RISCV Arch + LLVM
  - CompartOS (Almetary)
  - CheriOS(Esswood)
  - CHERI-RTOS (Xia)
  - Sail
  - Ibex / ETH Zurich / LowRISC
  - …

Summary

Fine-grained spatial and temporal memory safety guarantees for C/C++

Lightweight compartments

Safe bounded cross-compartment sharing



Strong attestation over compartment structure

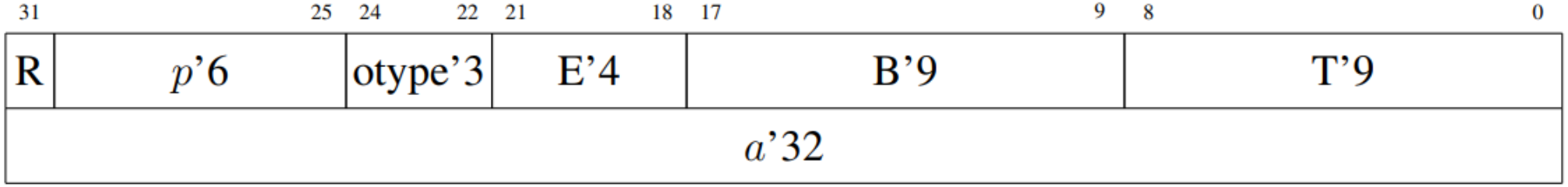
Any more questions, please ask in the GitHub Microsoft/CHERIoT-RTOS Discussions!
https://github.com/microsoft/cheriot-rtos/discussions/categories/q-a

Backup

Most codebases require very few changes

Microvium embedded JavaScript interpreter	TPM reference stack	FreeRTOS network stack	mBedTLS
• No changes	• No changes for memory safety • Small changes (<10LoC) for RISC-V • One line changed to run in a compartment	• No changes for memory safety • Annotations for cross-compartment calls • Explicit sealing and unsealing • Small changes (~100 LoC) to run without disabling interrupts for mutual exclusion	• No changes for memory safety • Small changes for compartmentalisation

Capability format



R a reserved bit, which is zero in the root capabilities (and hence all tagged capabilities), but may be set if untagged data is loaded into a register. In this case its value must be preserved. This is very important because memory copies are performed with capability load a store instructions in order to preserve the tag on any capabilities present, meaning these instructions must also faithfully copy arbitrary untagged data.

p a 6-bit compressed permissions field (see Section 7.13.1)

otype a 3-bit 'object type' used for sealing capabilities (see Section 7.13.2)

E a 4-bit exponent used for the bounds encoding (see Section 7.13.3)

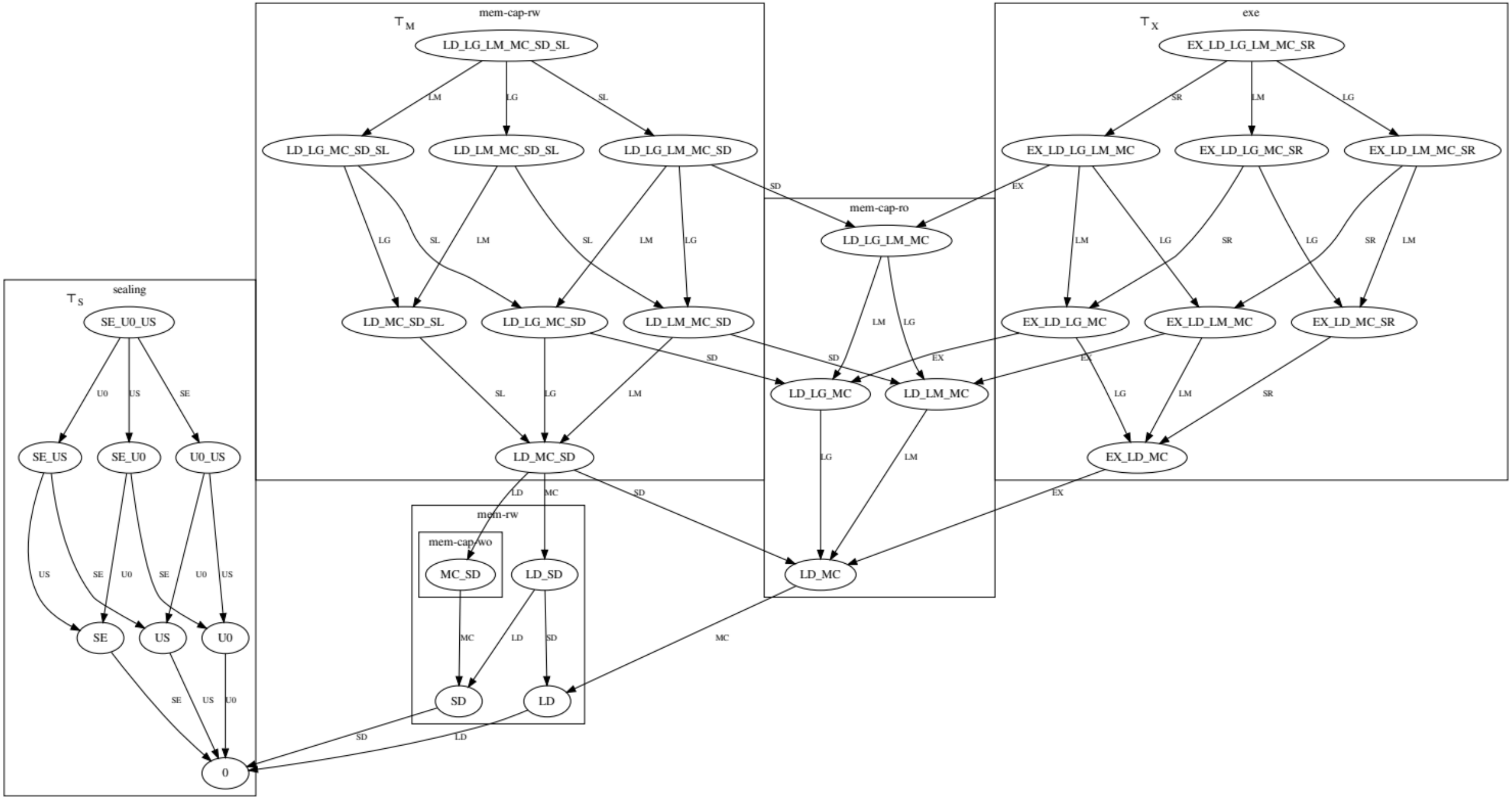
B a 9-bit base used for the bounds encoding (see Section 7.13.3)

T a 9-bit top used in the bounds encoding (see Section 7.13.3)

a the 32-bit address of the capability

Permission encoding

	5	4	3	2	1	0	
Memory cap-read-write:	GL	1	1	SL	LM	LG	Implicit: LD, MC, SD
Memory cap-read-only:	GL	1	0	1	LM	LG	Implicit: LD, MC
Memory cap-write-only:	GL	1	0	0	0	0	Implicit: SD, MC
Memory data-only:	GL	1	0	0	LD	SD	Implicit: None
Executable:	GL	0	1	SR	LM	LG	Implicit: EX, LD, MC
Sealing:	GL	0	0	U0	SE	US	Implicit: None


mem-cap-rw
$\top_M$
LD_LG_LM_MC_SD_SL
LM
LG
SL
LD_LG_MC_SD_SL
LD_LM_MC_SD_SL
LD_LG_LM_MC_SD
SD
LG
SL
LM
SL
LM
LG
LD_MC_SD_SL
LD_LG_MC_SD
LD_LM_MC_SD
SD
SD
SL
LG
LM
LD_MC_SD
LD
MC
SD
mem-rw
mem-cap-wo
MC_SD
LD_SD
MC
LD
SD
SD
LD
SD
LD
0
mem-cap-ro
LD_LG_LM_MC
LM
LG
LD_LG_MC
LD_LM_MC
LG
LM
LD_MC
MC
exe
$\top_X$
EX_LD_LG_LM_MC_SR
SR
LM
LG
EX_LD_LG_LM_MC
EX_LD_LG_MC_SR
EX_LD_LM_MC_SR
EX
LM
LG
SR
LG
SR
LM
EX_LD_LG_MC
EX_LD_LM_MC
EX_LD_MC_SR
EX
EX
LG
LM
SR
EX_LD_MC
EX
sealing
$\top_S$
SE_U0_US
U0
US
SE
SE_US
SE_U0
U0_US
US
SE
U0
SE
U0
US
SE
US
U0
SE
US
U0