Zephyr™ Project
Developer Summit
June 8-10, 2021 ▪ @ZephyrIoT

Demand Paging

When software is bigger than available memory

DANIEL LEUNG
INTEL CORPORATION

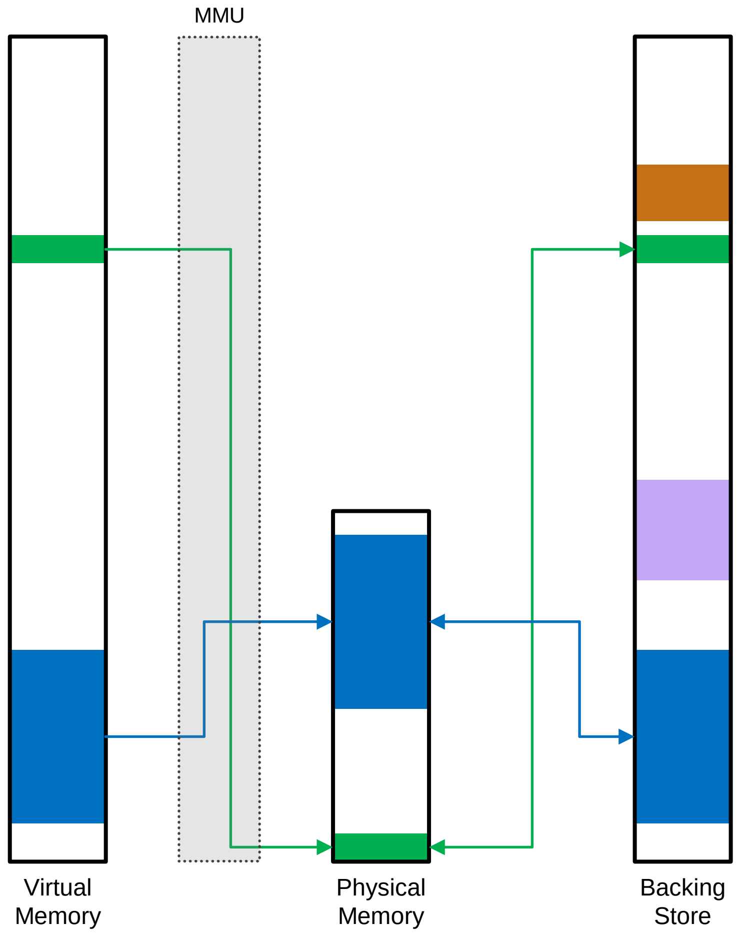
“640K ought to be enough for anybody”

Agenda

- What is Demand Paging
- Why Demand Paging
- Demand Paging in Zephyr
- Current Status

What is Demand Paging

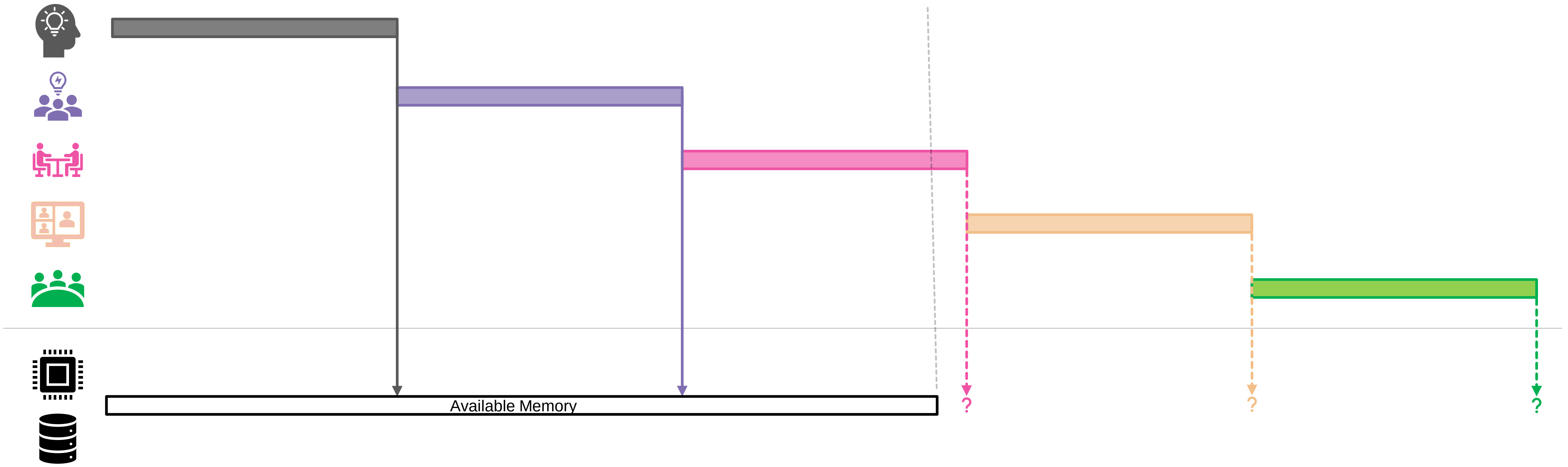
- Virtual memory management scheme where a memory page is brought into physical memory if access is needed
  - Application has access to more usable memory
  - Need MMU for translation between physical and virtual addresses
  - Requires Backing Store to store page not in physical memory
    - Usually storage with lower cost than memory hardware (e.g. SRAM)
    - Modified page will need to be written back into backing store



MMU – Memory Management Unit

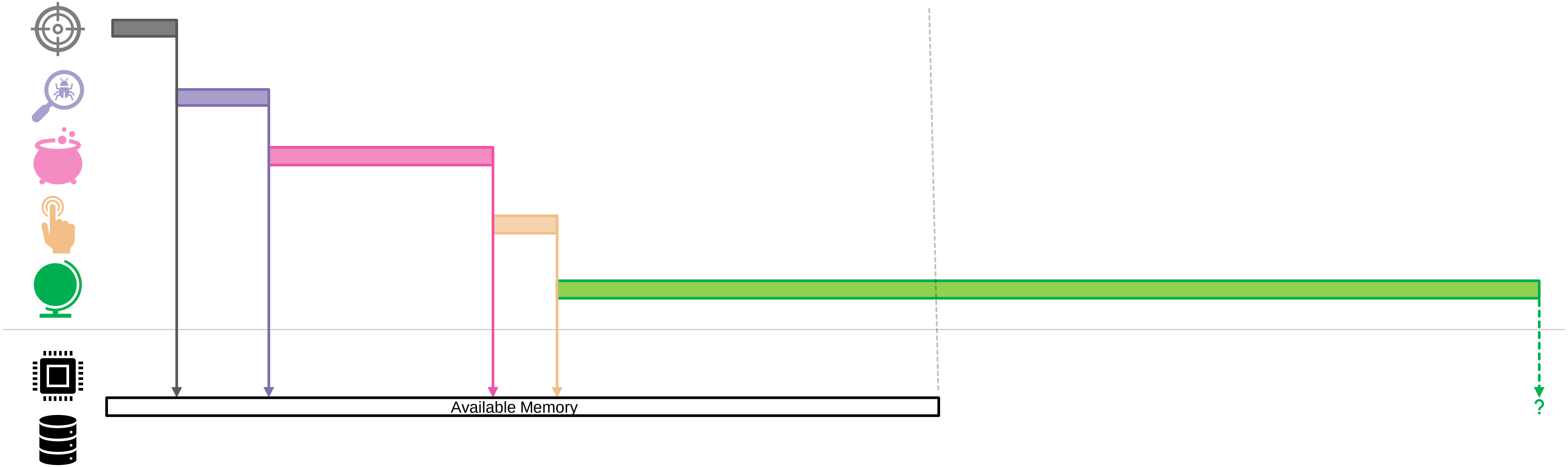
Why Demand Paging

- More code, more data → need more memory
  - More features and complex ideas



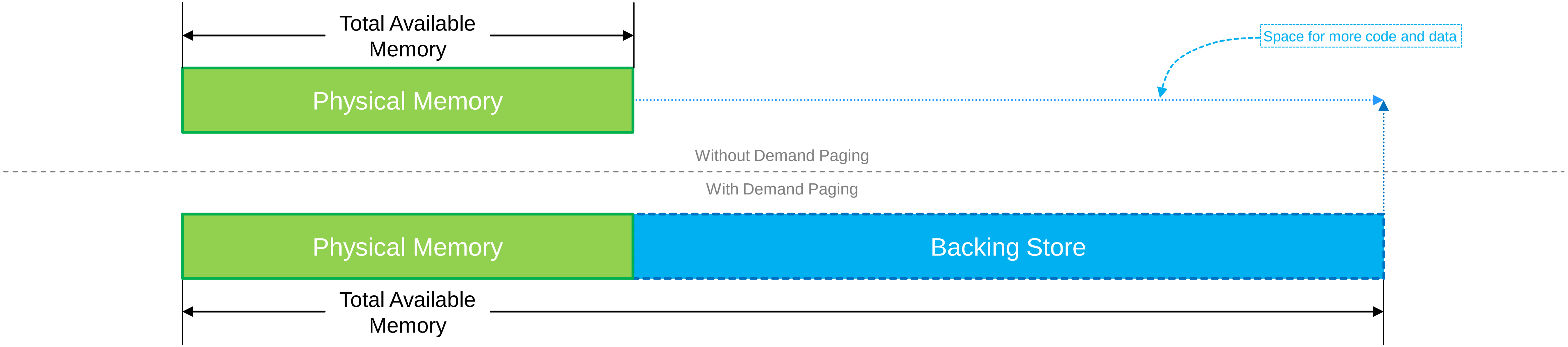
Why Demand Paging

- Consolidation → do more with one device
  - More and more code and data



Why Demand Paging

- Demand paging gives the application a larger address space for code and data



… basically… that's it.

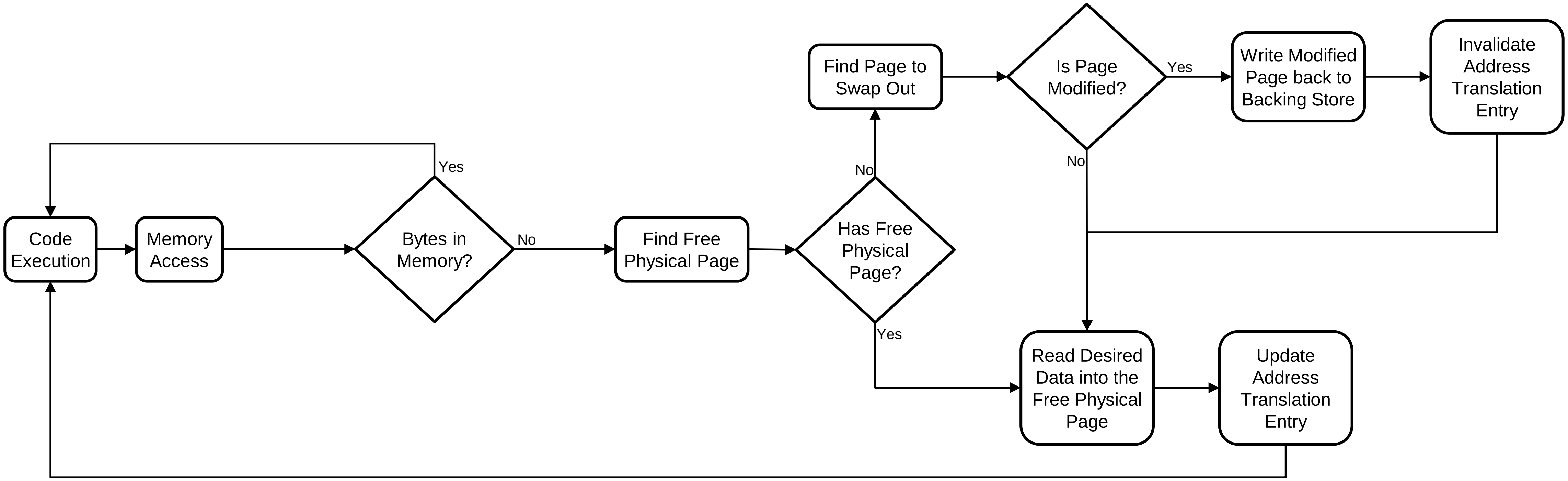
Why Demand Paging

- But there are drawbacks…
  - Requires hardware support for address translation (e.g. MMU)
    - Code execution and data access are in virtual address space
  - Additional hardware as backing store
  - Latency due to swapping memory pages between physical memory and backing store
    - If page is already in physical memory → Negligible
    - If not, need to copy from usually much slower backing store to physical memory.
      - More if no free physical pages available and need to copy data back into backing store to free up a physical page.

Demand Paging in Zephyr

Zephyr™ Project
Developer Summit

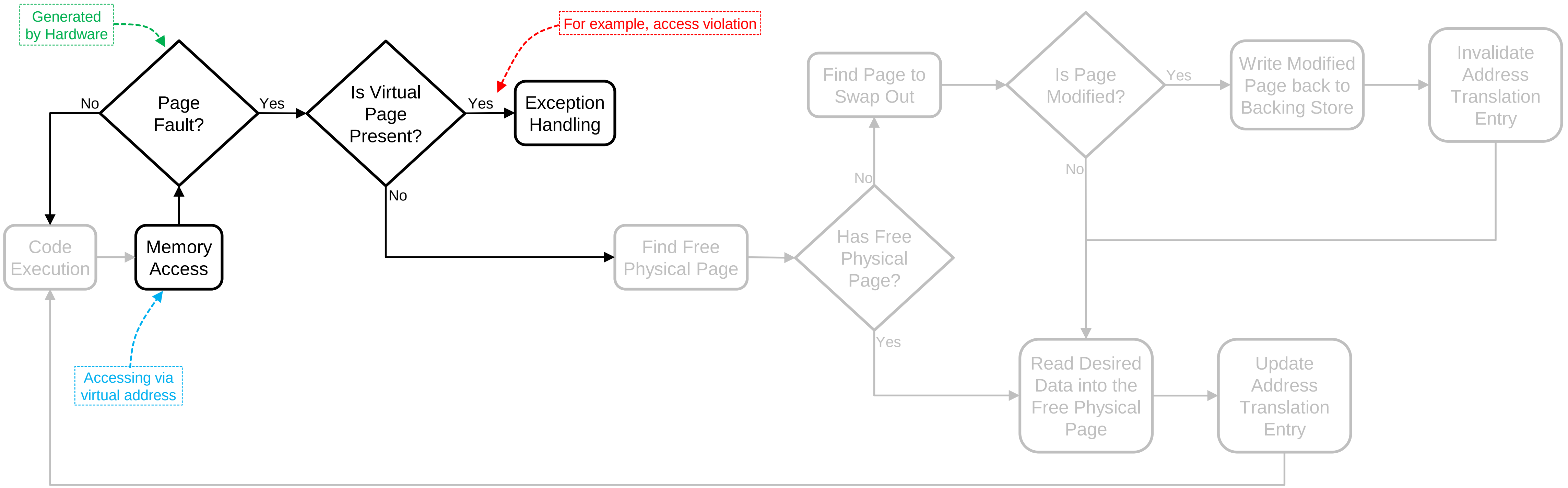
- Conceptually…



Code Execution
Memory Access
Bytes in Memory?
Yes
No
Find Free Physical Page
Has Free Physical Page?
No
Yes
Find Page to Swap Out
Is Page Modified?
Yes
No
Write Modified Page back to Backing Store
Invalidate Address Translation Entry
Read Desired Data into the Free Physical Page
Update Address Translation Entry

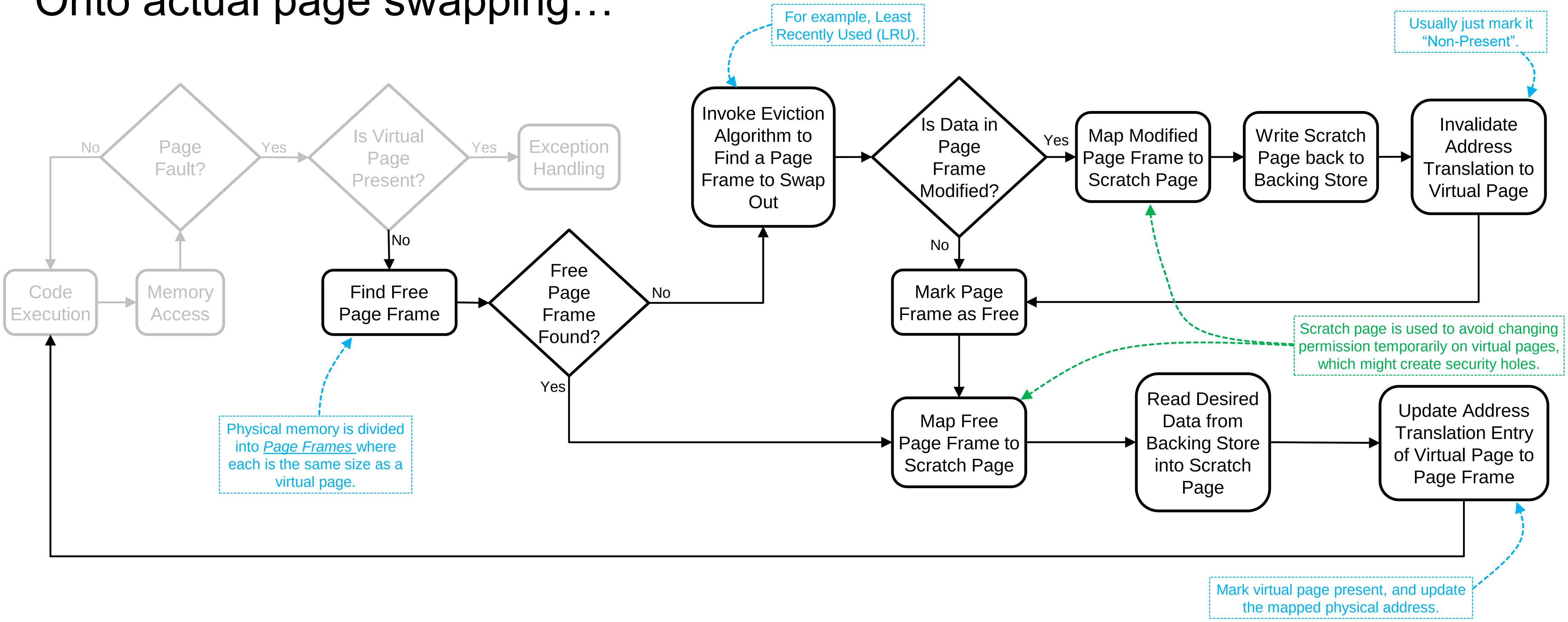
Demand Paging in Zephyr

- How Zephyr handle when bytes not in memory…


Generated by Hardware
Page Fault?
No
Yes
Is Virtual Page Present?
Yes
For example, access violation
Exception Handling
No
Code Execution
Memory Access
Accessing via virtual address
Find Free Physical Page
Has Free Physical Page?
No
Yes
Find Page to Swap Out
Is Page Modified?
Yes
No
Write Modified Page back to Backing Store
Invalidate Address Translation Entry
Read Desired Data into the Free Physical Page
Update Address Translation Entry

Demand Paging in Zephyr

- Onto actual page swapping…


Code Execution
Memory Access
Page Fault?
No
Yes
Is Virtual Page Present?
Yes
Exception Handling
No
Find Free Page Frame
Free Page Frame Found?
No
Yes
Invoke Eviction Algorithm to Find a Page Frame to Swap Out
Is Data in Page Frame Modified?
Yes
No
Map Modified Page Frame to Scratch Page
Write Scratch Page back to Backing Store
Invalidate Address Translation to Virtual Page
Mark Page Frame as Free
Map Free Page Frame to Scratch Page
Read Desired Data from Backing Store into Scratch Page
Update Address Translation Entry of Virtual Page to Page Frame

For example, Least Recently Used (LRU).

Usually just mark it "Non-Present".

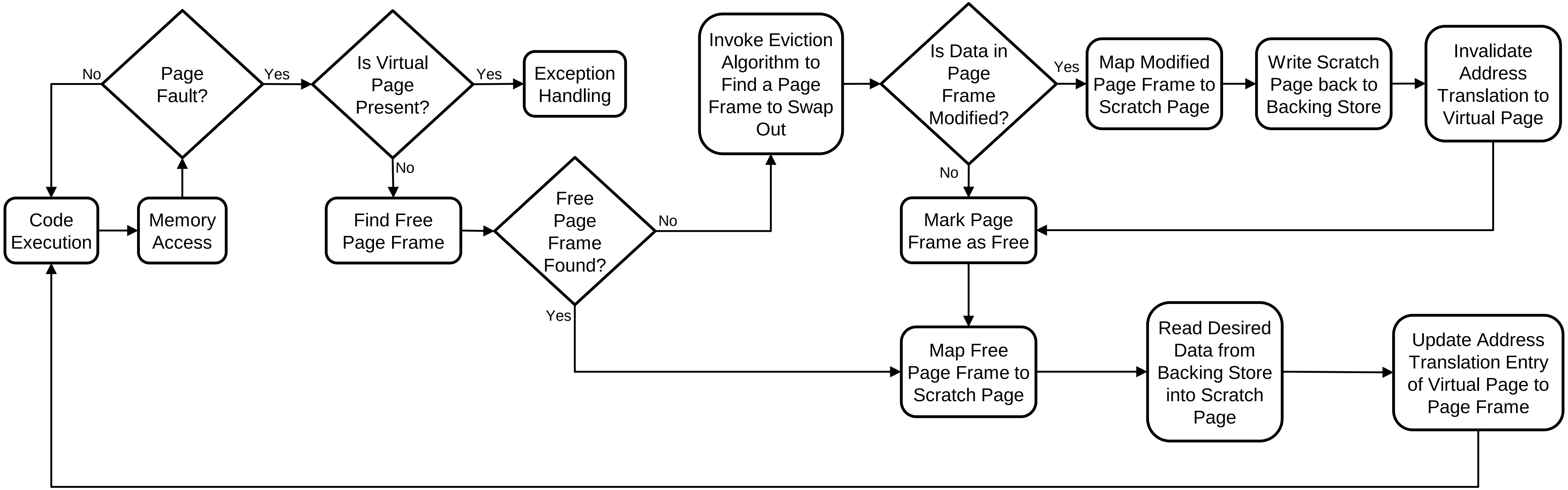
Scratch page is used to avoid changing permission temporarily on virtual pages, which might create security holes.

Physical memory is divided into *Page Frames* where each is the same size as a virtual page.

Mark virtual page present, and update the mapped physical address.

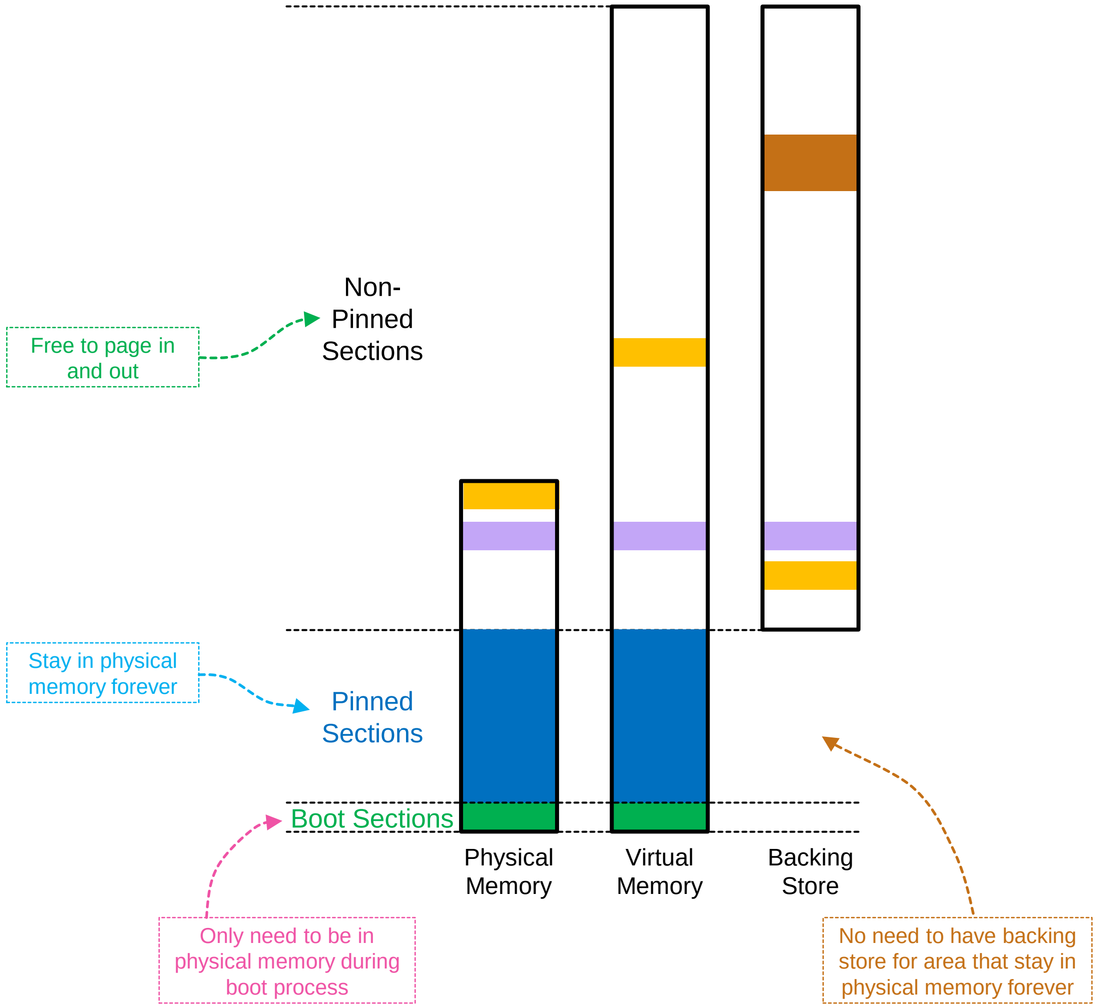
Demand Paging in Zephyr

- Overall…


Page Fault?
No
Yes
Is Virtual Page Present?
Yes
Exception Handling
No
Code Execution
Memory Access
Find Free Page Frame
Free Page Frame Found?
No
Yes
Invoke Eviction Algorithm to Find a Page Frame to Swap Out
Is Data in Page Frame Modified?
Yes
No
Map Modified Page Frame to Scratch Page
Write Scratch Page back to Backing Store
Invalidate Address Translation to Virtual Page
Mark Page Frame as Free
Map Free Page Frame to Scratch Page
Read Desired Data from Backing Store into Scratch Page
Update Address Translation Entry of Virtual Page to Page Frame

Demand Paging in Zephyr

- But wait… what if the eviction algorithm code is paged out?!
  - Results in a page fault within a page fault, within another page fault and so on…
  - Solution is to PIN the code in memory…
    - Necessary code and data get put into particular linker sections.
    - Everything in these sections are *pinned* in memory and will not be paged out.


Free to page in and out
Non-Pinned Sections
Stay in physical memory forever
Pinned Sections
Boot Sections
Physical Memory
Virtual Memory
Backing Store
Only need to be in physical memory during boot process
No need to have backing store for area that stay in physical memory forever

Current Status

- On architecture,
  - Only x86-32 has the architecture code to support demand paging.
- On eviction algorithm,
  - Only has a sample NRU (Not Recently Used) eviction algorithm in tree for testing.
- On backing store implementation,
  - One using memslab is used as demo and for testing.

Zephyr™ Project
Developer Summit

Questions?

Zephyr™ Project

Developer Summit

June 8-10, 2021 ▪ @ZephyrIoT