OpenCL: Programming Model & Philosophy

GPU Programming in Toronto

AJ Guillon

October 3, 2013

About Me

- B.Sc University of Toronto
  - Advanced OS, DBMS courses
  - Advanced mathematics and computer science theory
- Skills = Computer Science ∩ Software Engineering ∩ Computer Architecture
- Extensive parallel programming experience
  - Wait-free algorithm design

    Means that threads do not interfere with each other
  - Parallelization of hard parallel problems

    P = NC?
  - Deep understanding of low-level details
- OpenCL Expert
  - Started in 2008 (before any implementations were out)
  - Currently writing critiques of OpenCL 2.0

About Me

- C++ expert
  - Still exploring parts of C++11
  - Template Meta Programming
  - Starting to get involved with standards group

    SG7: Compile-time reflection
- OpenCL middleware developer
  - Goal: provide useful and efficient OpenCL abstractions

    You don't want to program in OpenCL directly!
  - I'm currently on my 7th iteration

    Finding the right abstractions has been hard!
  - Goal is a dual-licensed OpenCL C++ library

    Something like Threading Building Blocks (TBB), not like BOLT or Thrust

Work Experience

- Co-founder of two startup companies
  - Most recently a rock mechanics and technology company

    Built internal OpenCL tools and libraries
- Trying out life as a consultant now
  - Training, algorithm development, code reviews, etc.

    Previously offered 2-day hands-on workshop for limited audience at UHN
  - Starting another company to build OpenCL middleware

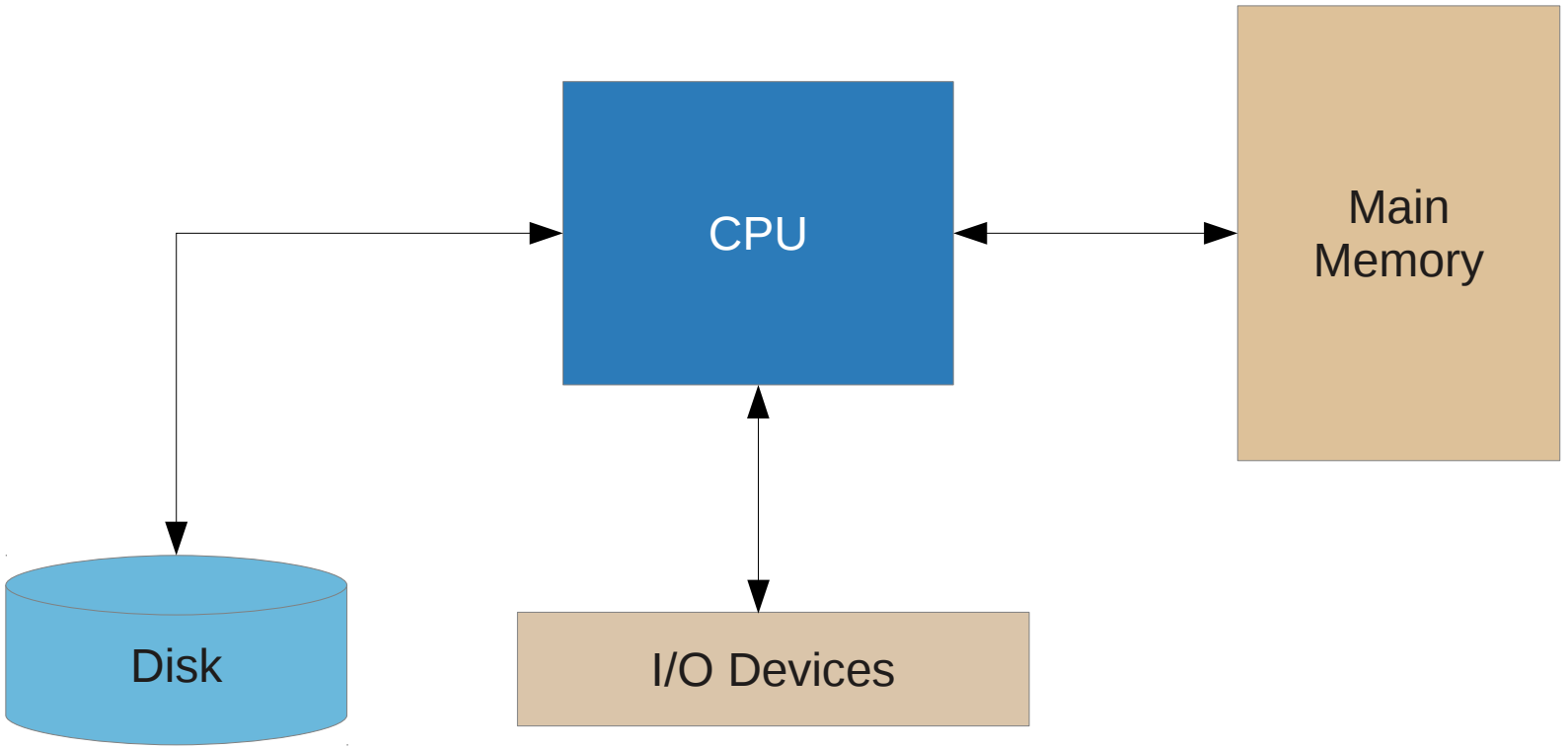
    Goal: mixed open-source and proprietary software company.
  - Also doing R&D on a new approach to parallel programming
- Seeking consulting work

http://www.ajguillon.com

aj@ajguillon.com

Heterogeneous Computing

Traditional Computing



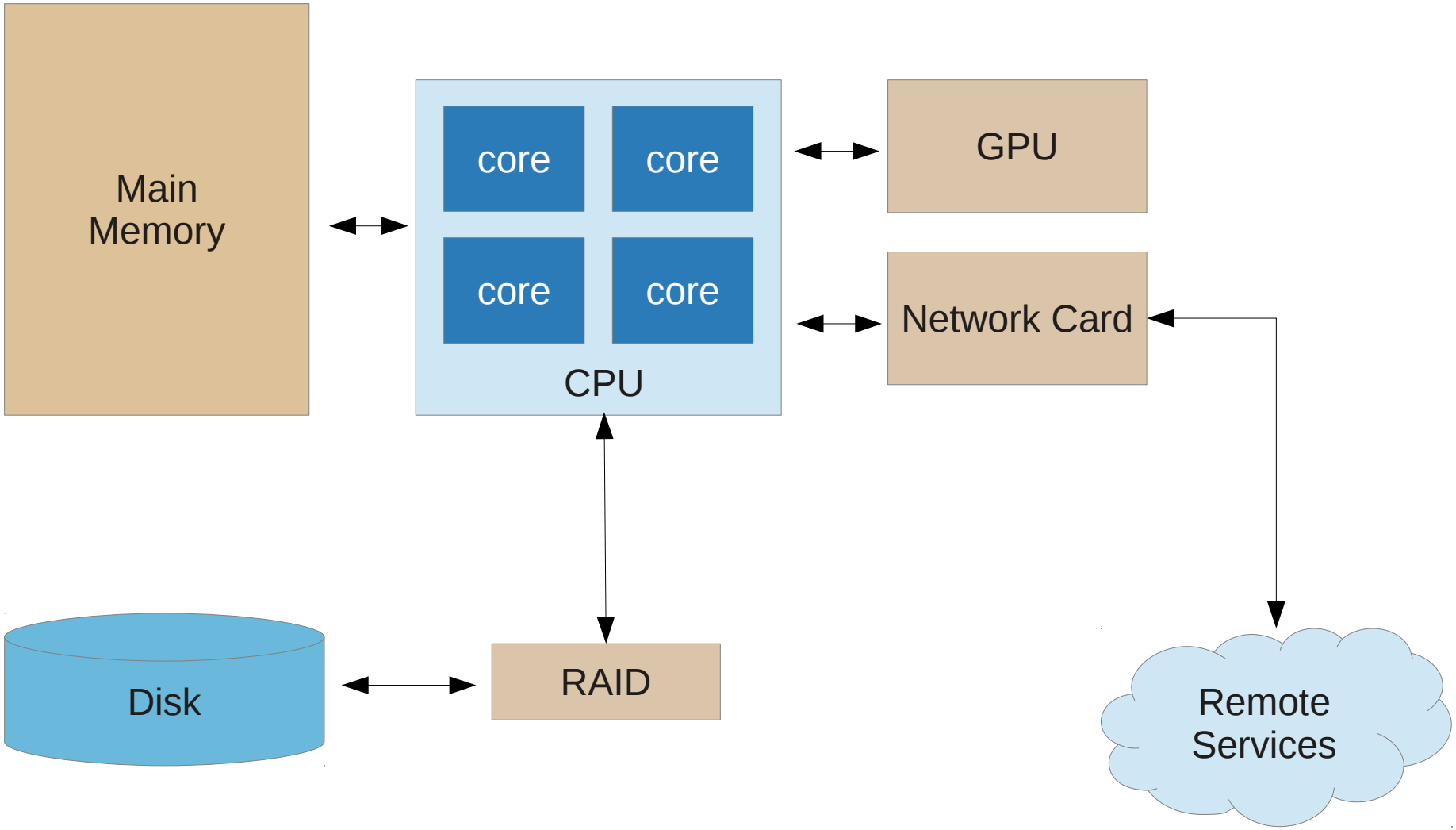
Traditional Computing: Notes

- The CPU was quite overloaded
  - Operating system runs here
  - Programs run here
- Hardware trends alleviated pressure on the CPU
  - Offset rendering to specialized hardware (GPU)

    GPUs became more and more programmable with each generation.
  - DMA freed up I/O management
  - RAID controllers have processors too
  - Specialized devices are just plain faster

    But over time they may evolve to be highly programmable.

Computing Today

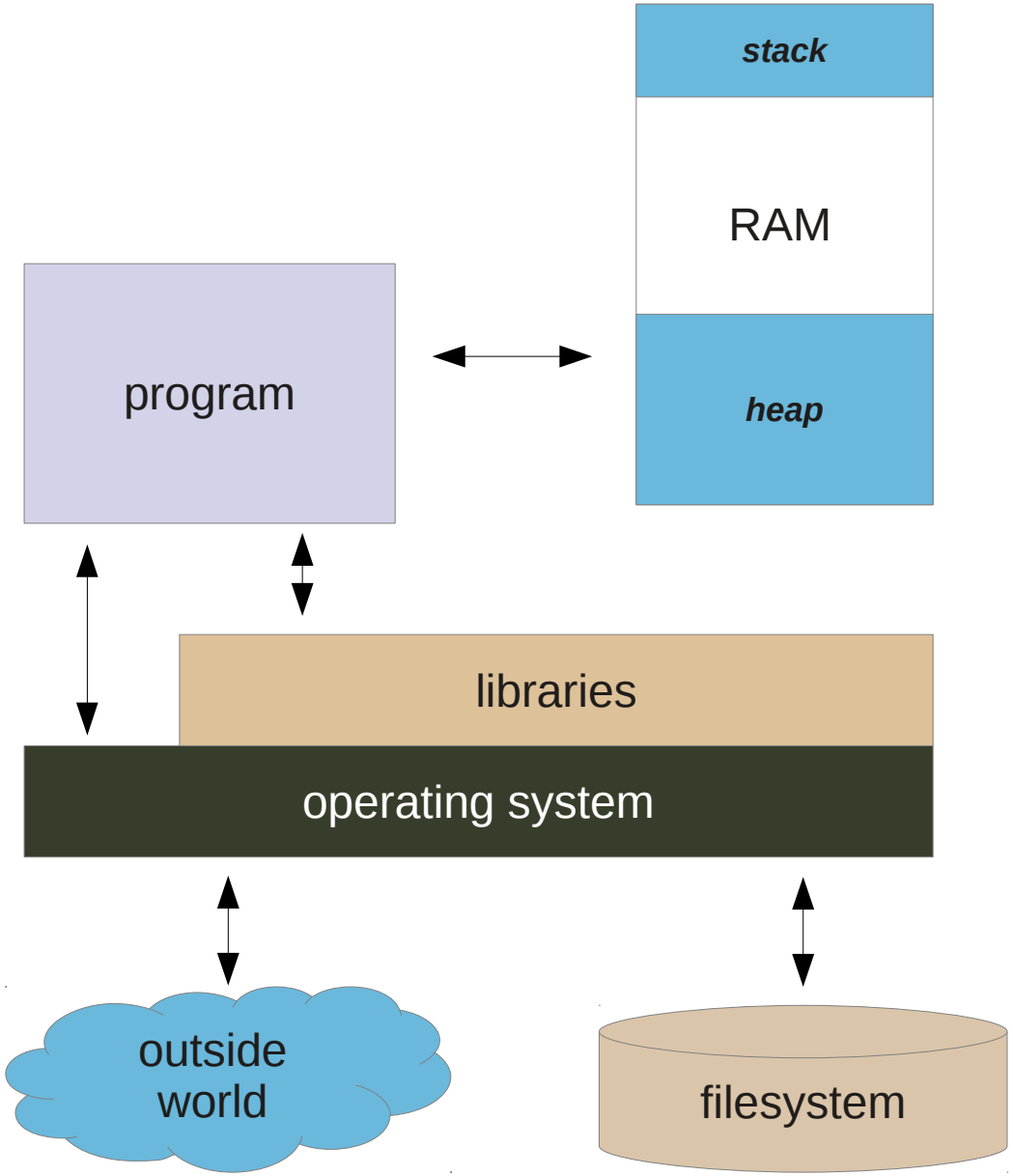


Computing Today: Notes

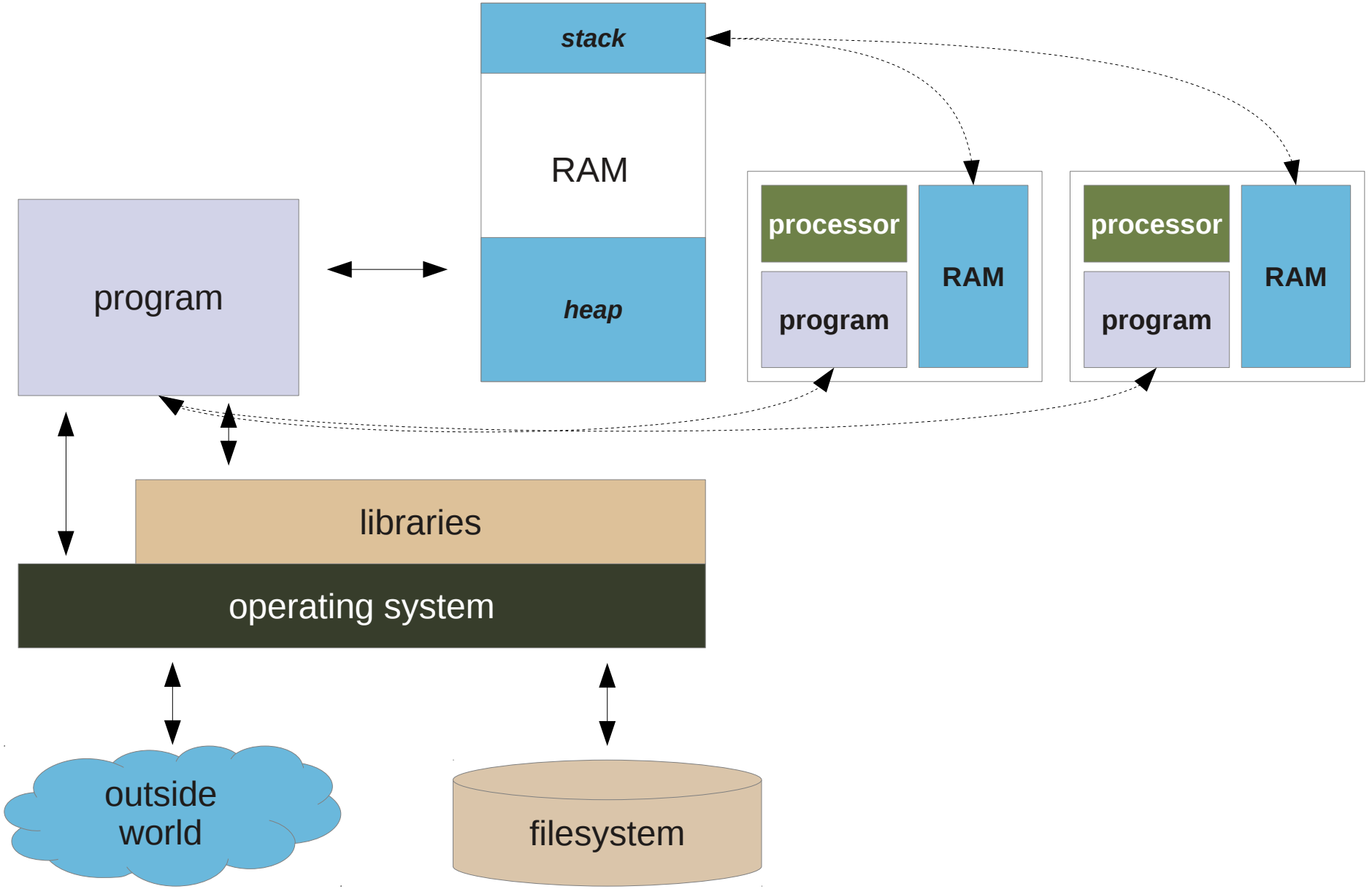
- Multi-core CPU
  - Operating system runs here
  - Programs run here
- Attached devices are programmable
  - Modern GPUs are programmable and very fast

    With the right workload, faster than the CPU!
  - CPU still tells the devices what to do
  - Devices and CPU all have different ISAs
- Remote services
  - Might offload computation to a remote cluster
  - Data might be fetched and stored to remote locations

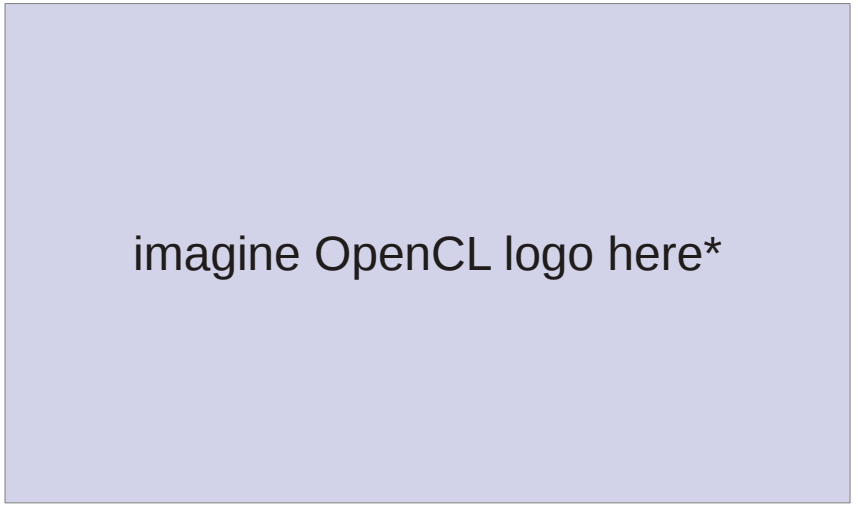
Traditional Software



Heterogeneous Software



OpenCL Philosophy



** Apple provides a horribly complicated legal framework to use the OpenCL logo, so let's use our imagination!*

My Favorite Quote

“OpenCL, however, is an unusually complex parallel programming standard. It has to be. I am aware of no other parallel programming model that addresses such a wide array of systems: GPUs, CPUs, FPGAs, embedded processors, and combinations of these systems. OpenCL is also complicated by the goals of its creators. You see, in creating OpenCL, we decided the best way to impact the industry would be to create a programming model for the performance-oriented programmer wanting full access to the details of the system. Our reasoning was that, over time, high-level models would be created to map onto OpenCL. By creating a common low-level target for these higher level models, we'd enable a rich marketplace of ideas and programmers would win. OpenCL, therefore, doesn't give you many abstractions to make your programming job easier. You have to do all that work yourself.”

– Tim Mattson, Principal Engineer, Intel Corp.

OpenCL Philosophy

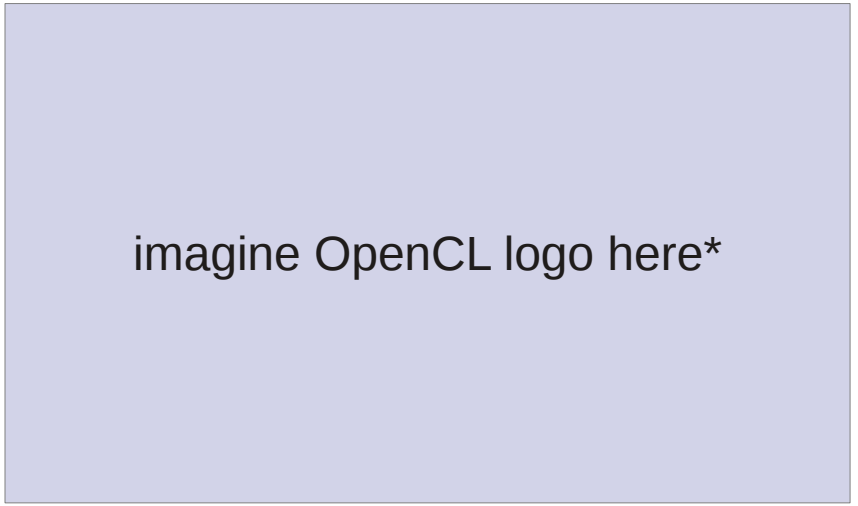
- OpenCL is a low-level standard
  - You really don't want to use it directly
- High-level libraries will provide abstractions
  - We don't know what they will be yet
  - OpenCL allows us to search for them

    … and maintain program portability
- Enable an ecosystem of software and hardware

OpenCL Philosophy

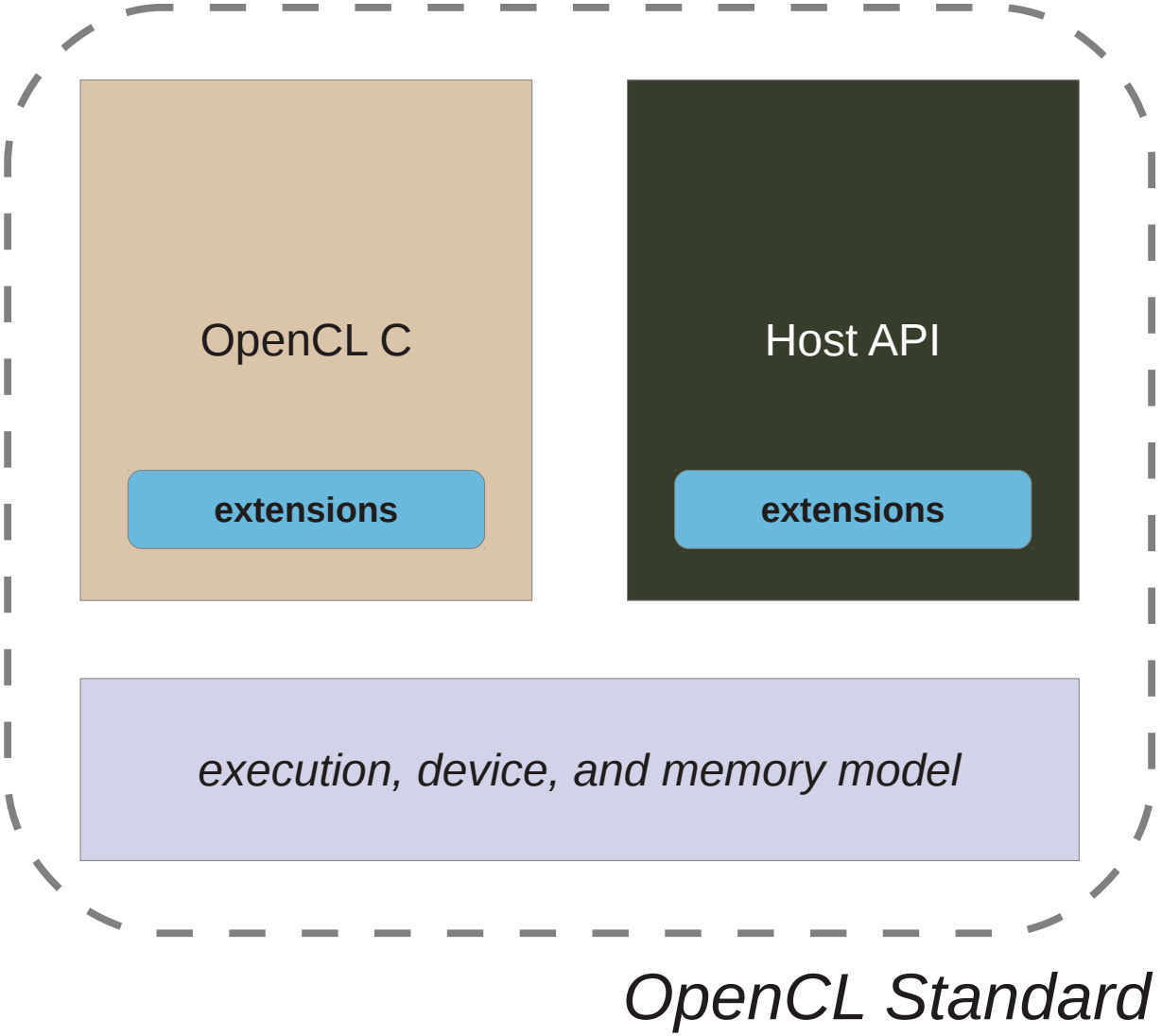
… to be continued

OpenCL 1.2 Model



** Apple provides a horribly complicated legal framework to use the OpenCL logo, so let's use our imagination!*

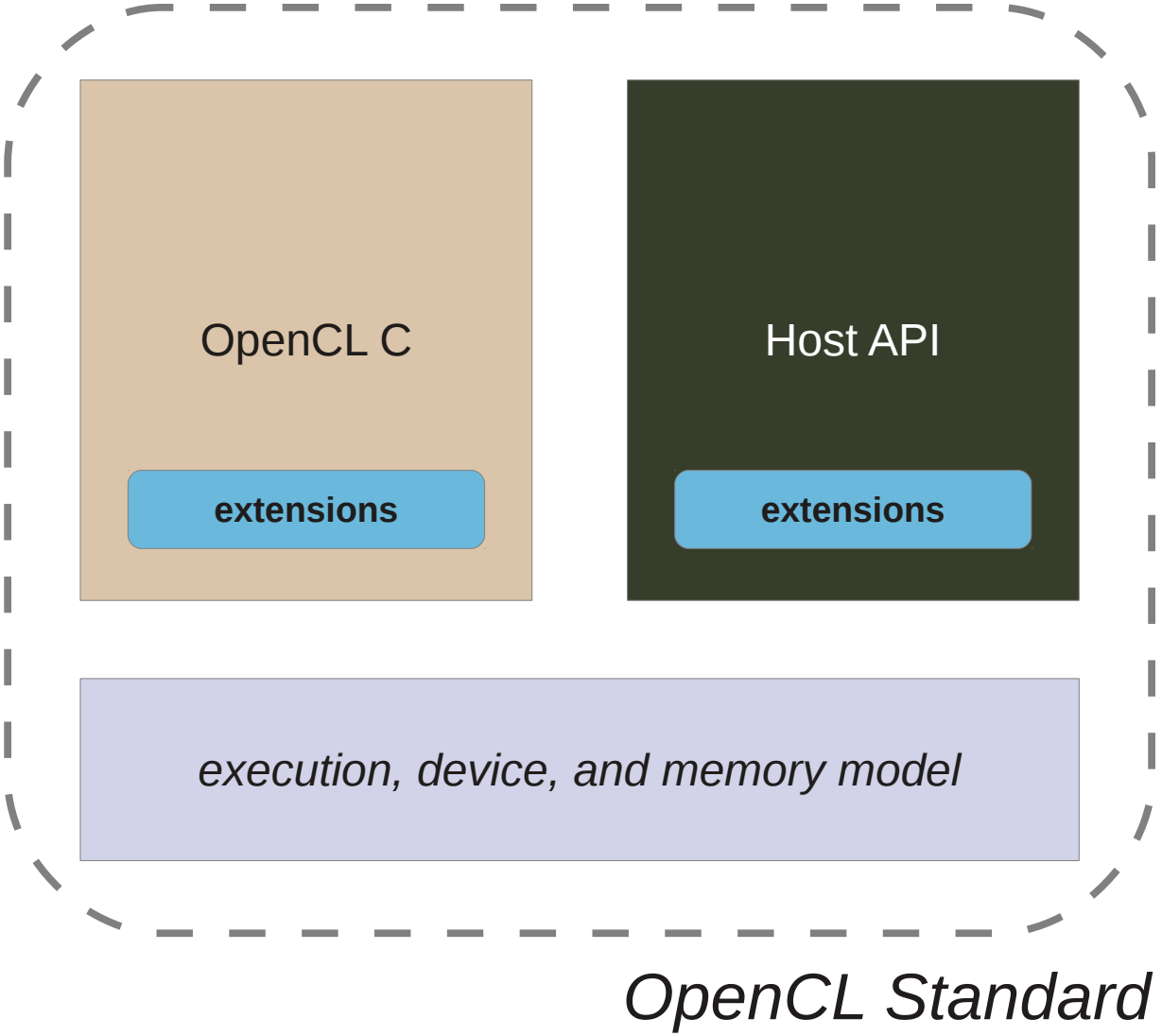
OpenCL Components



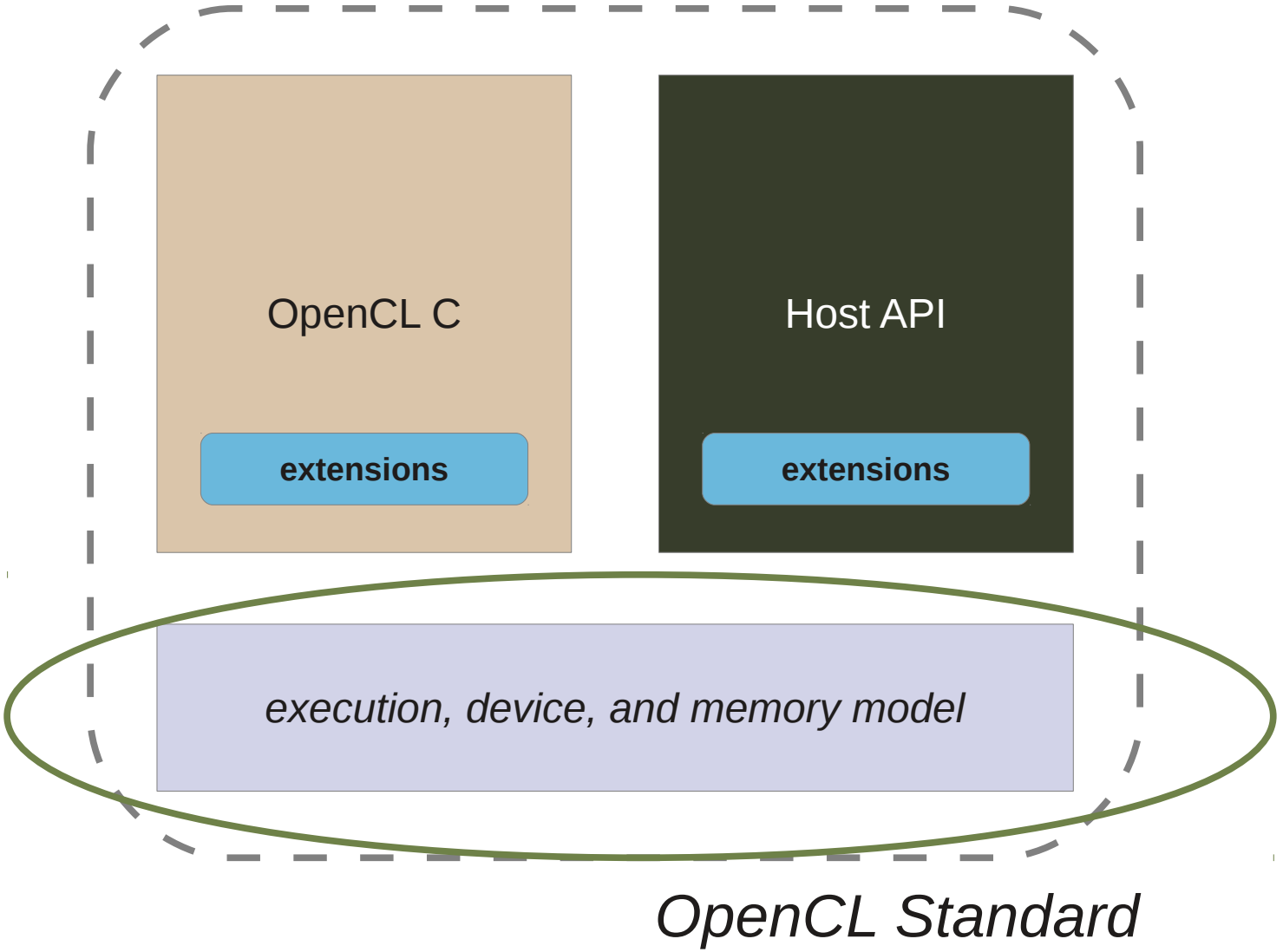
OpenCL Standard

- Core specification
  - Everything supports this
  - Changes from version to version
- Extensions
  - Add functionality to OpenCL
- Embedded profile
  - Subset of standard for embedded devices
- Custom devices
  - Don't support programs
  - Fixed-function hardware
- Many versions
  - 1.0, 1.1, 1.2, 2.0 (provisional)

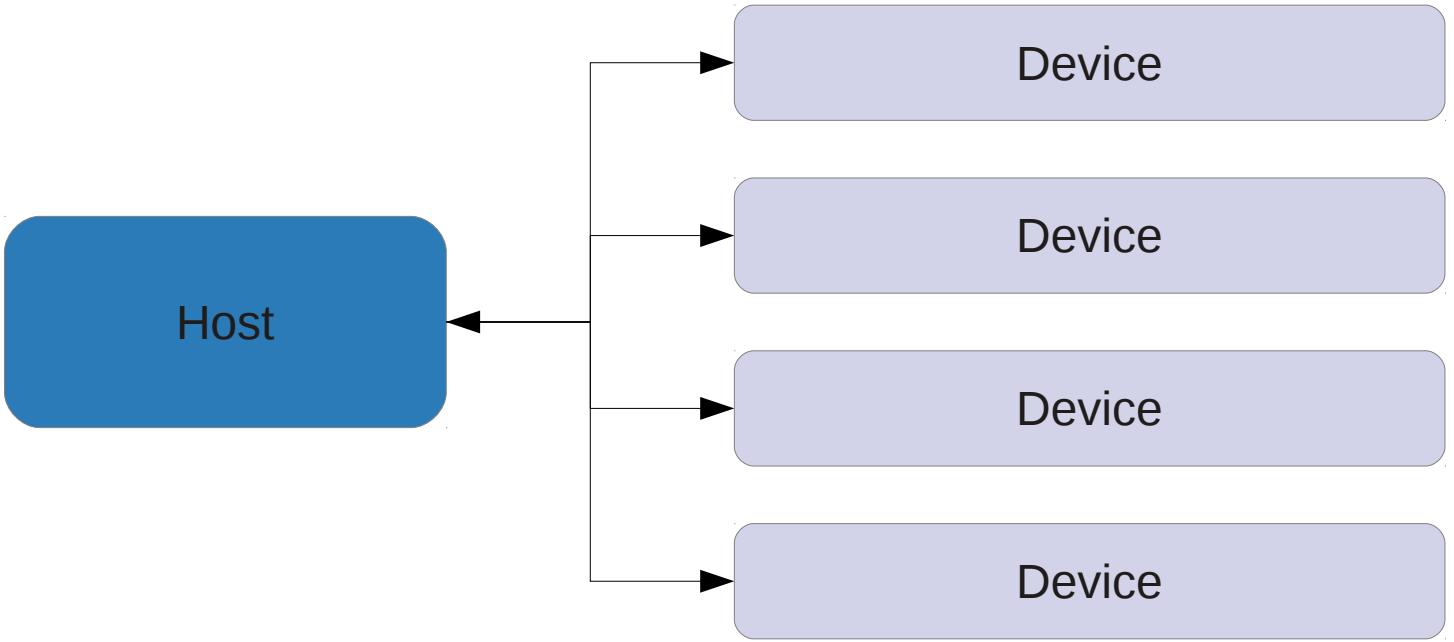
OpenCL Components



OpenCL Components

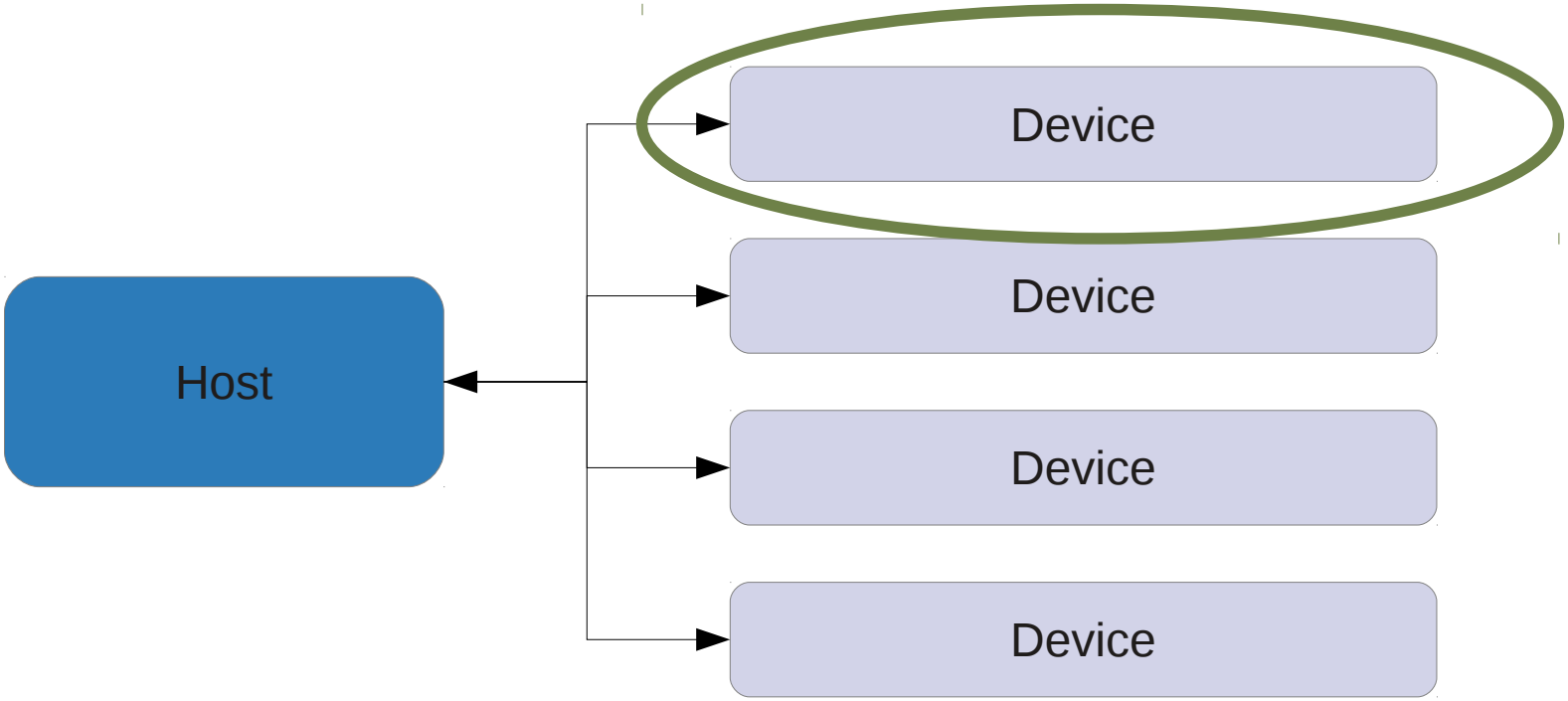


The OpenCL Model

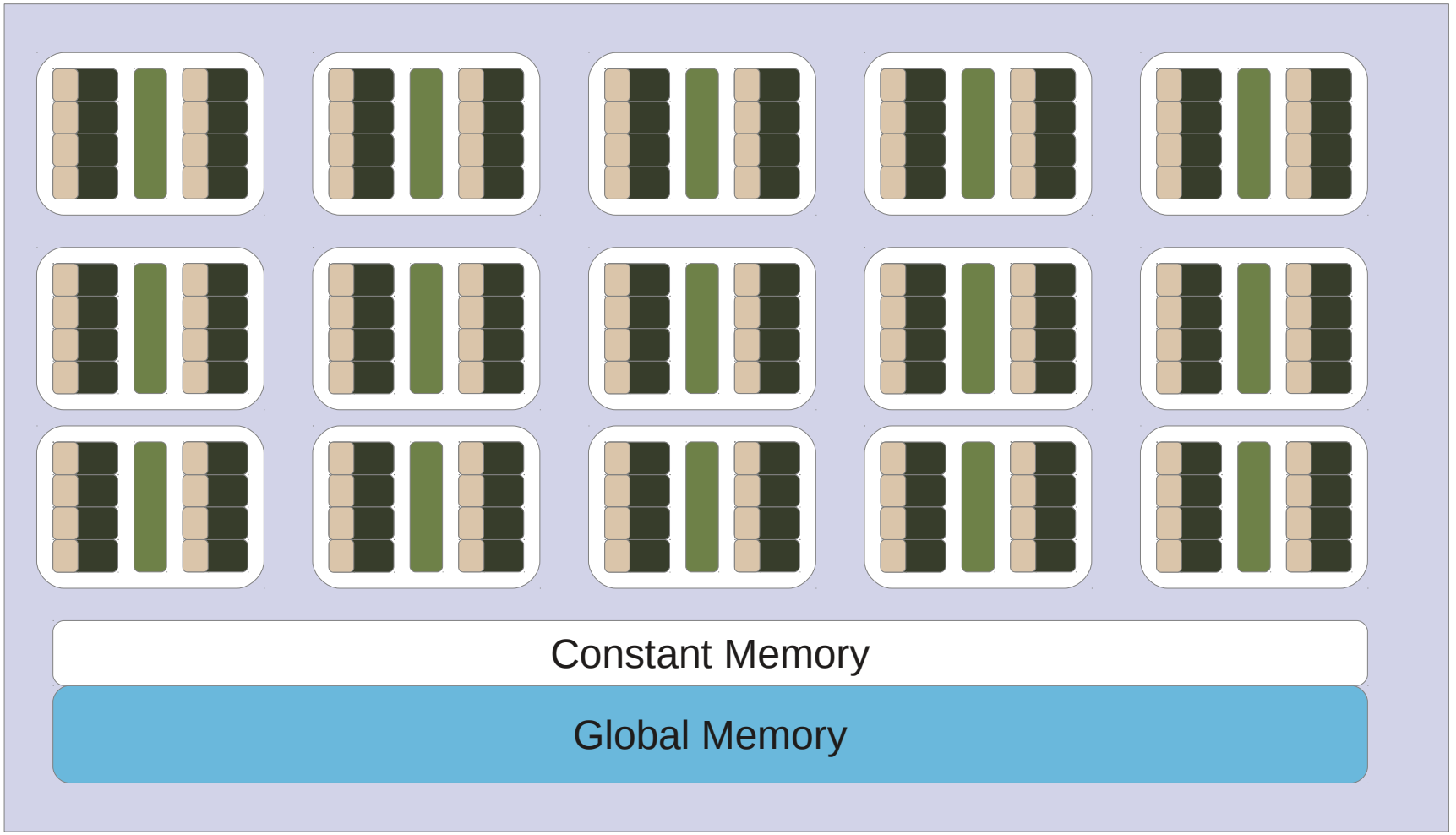


Quick Tour!

Let's zoom in....

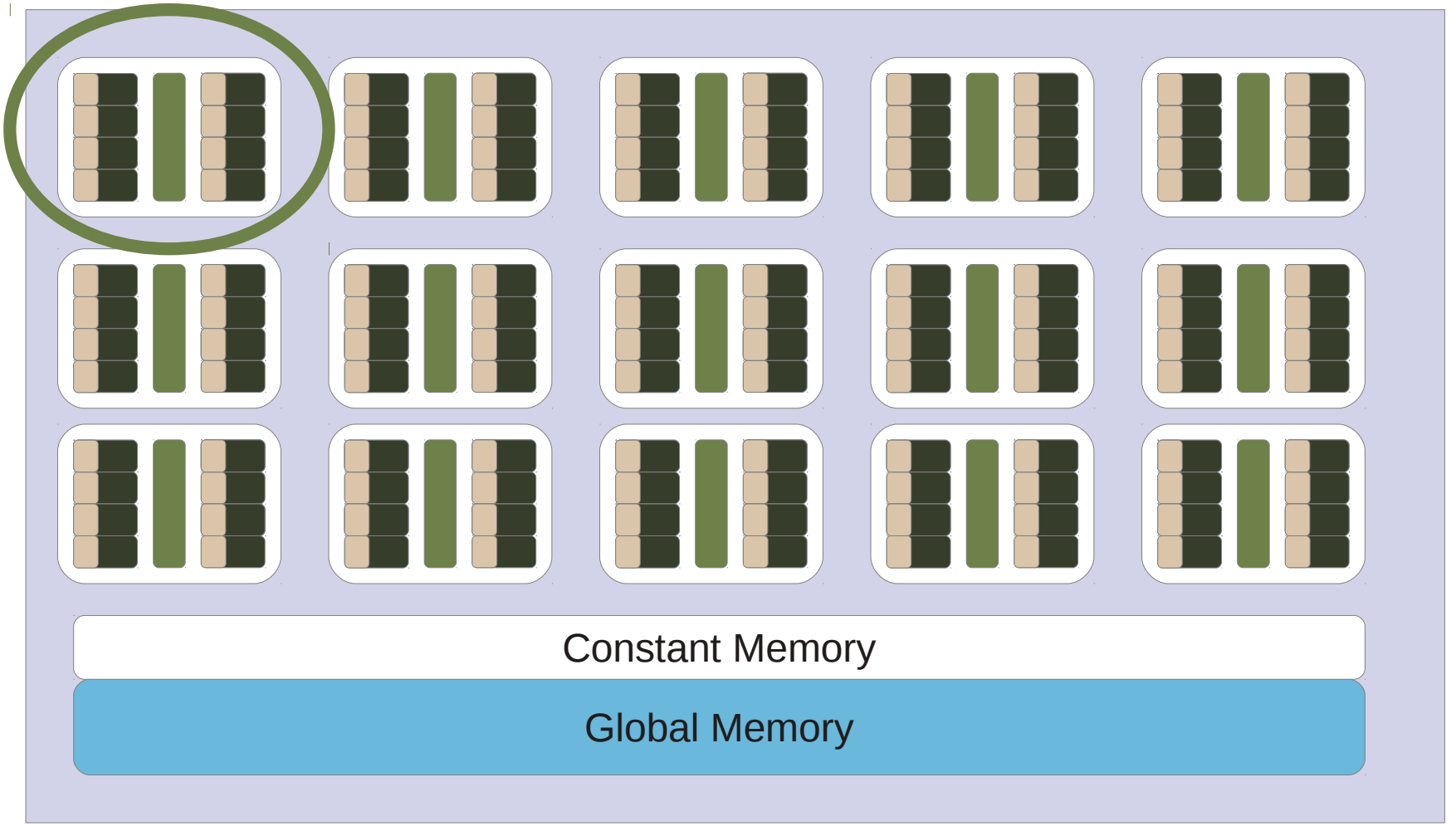


Inside the Device

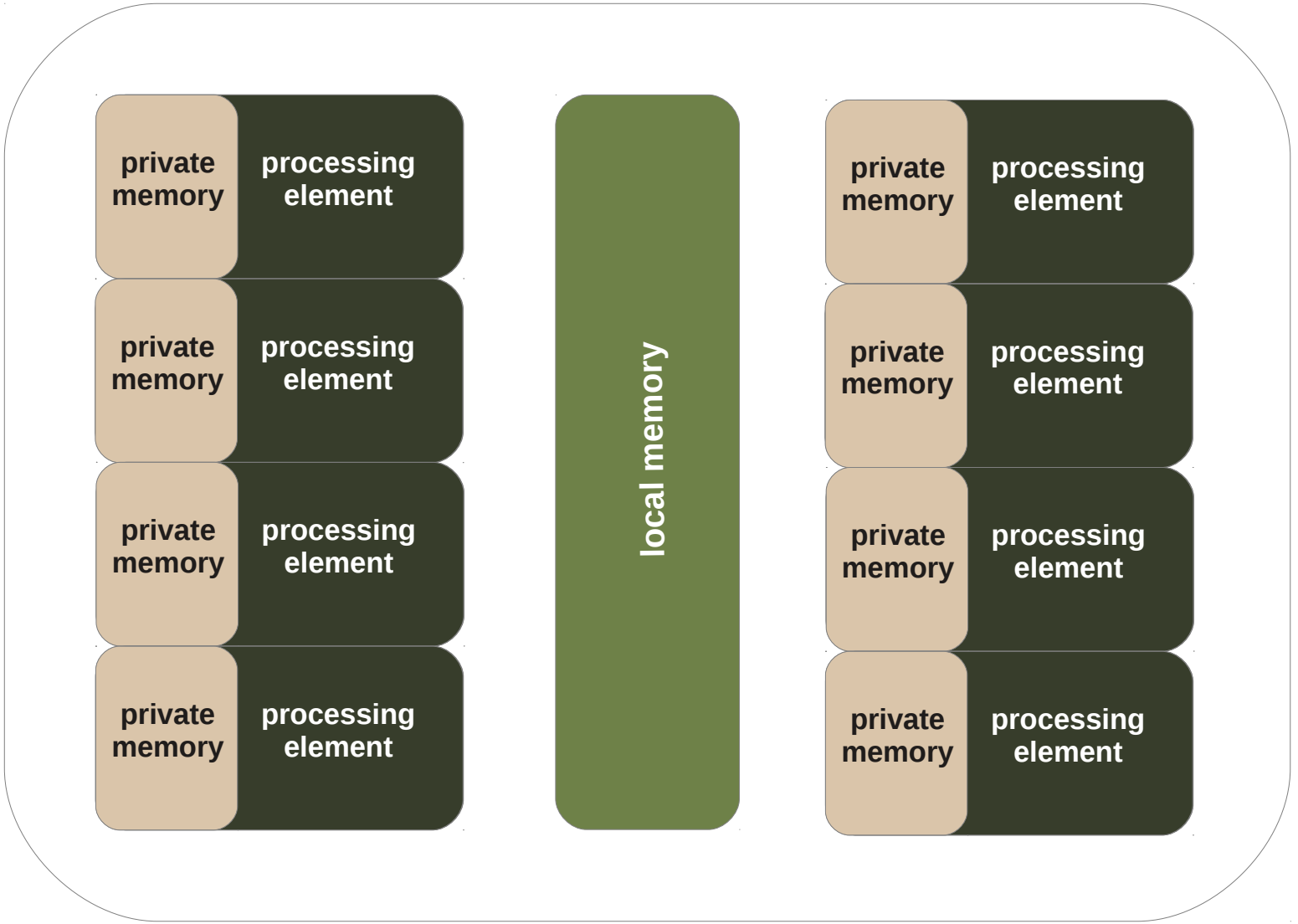


Inside the Device

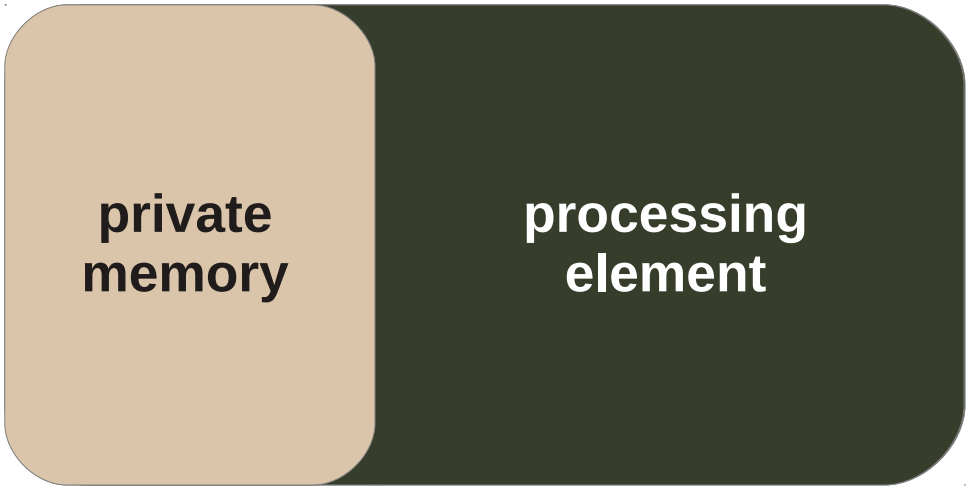
A compute unit (CU)



Inside the Compute Unit (CU)



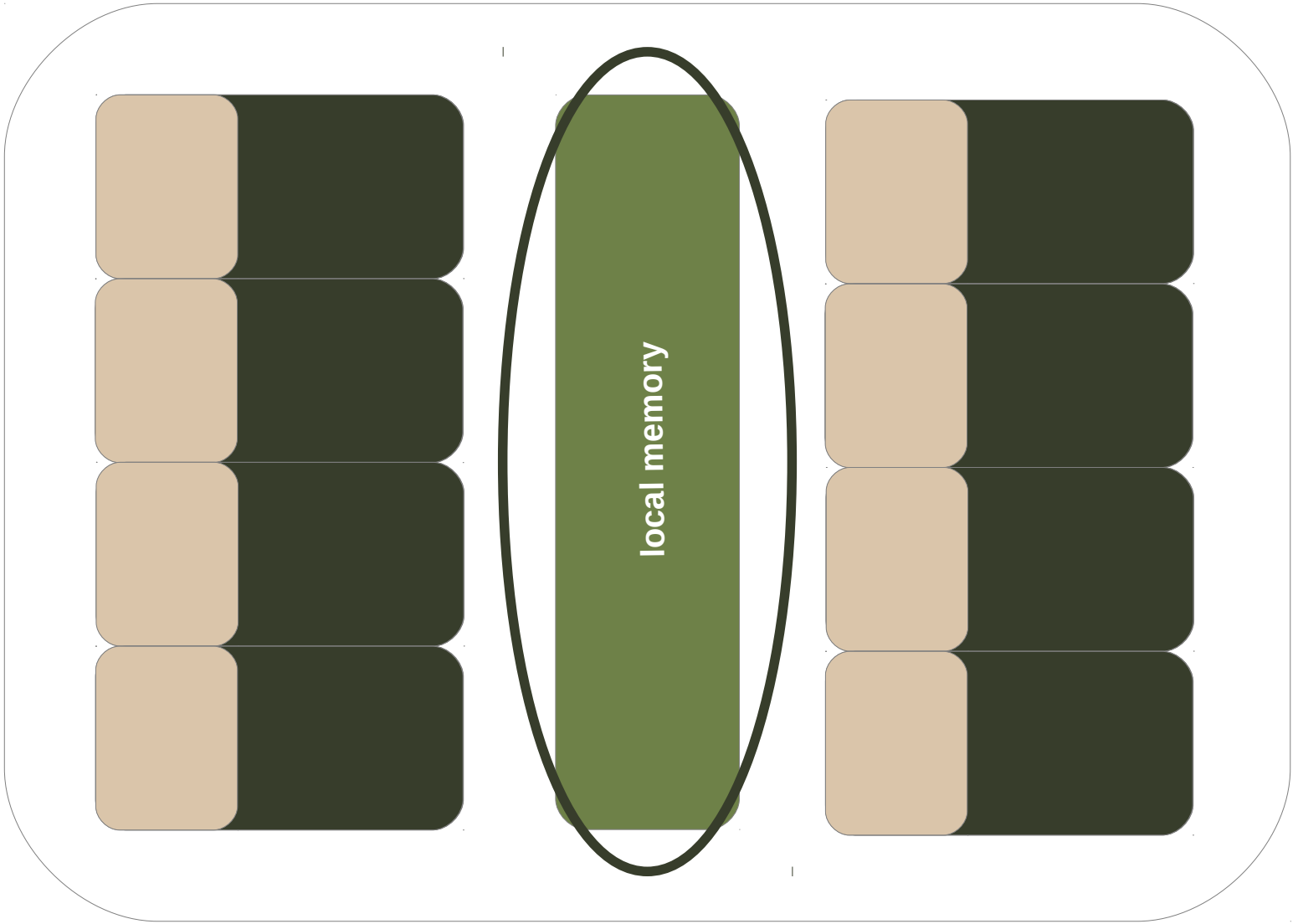
Inside the Processing Element (PE)



The *processing element (PE)* does all computation on the device.

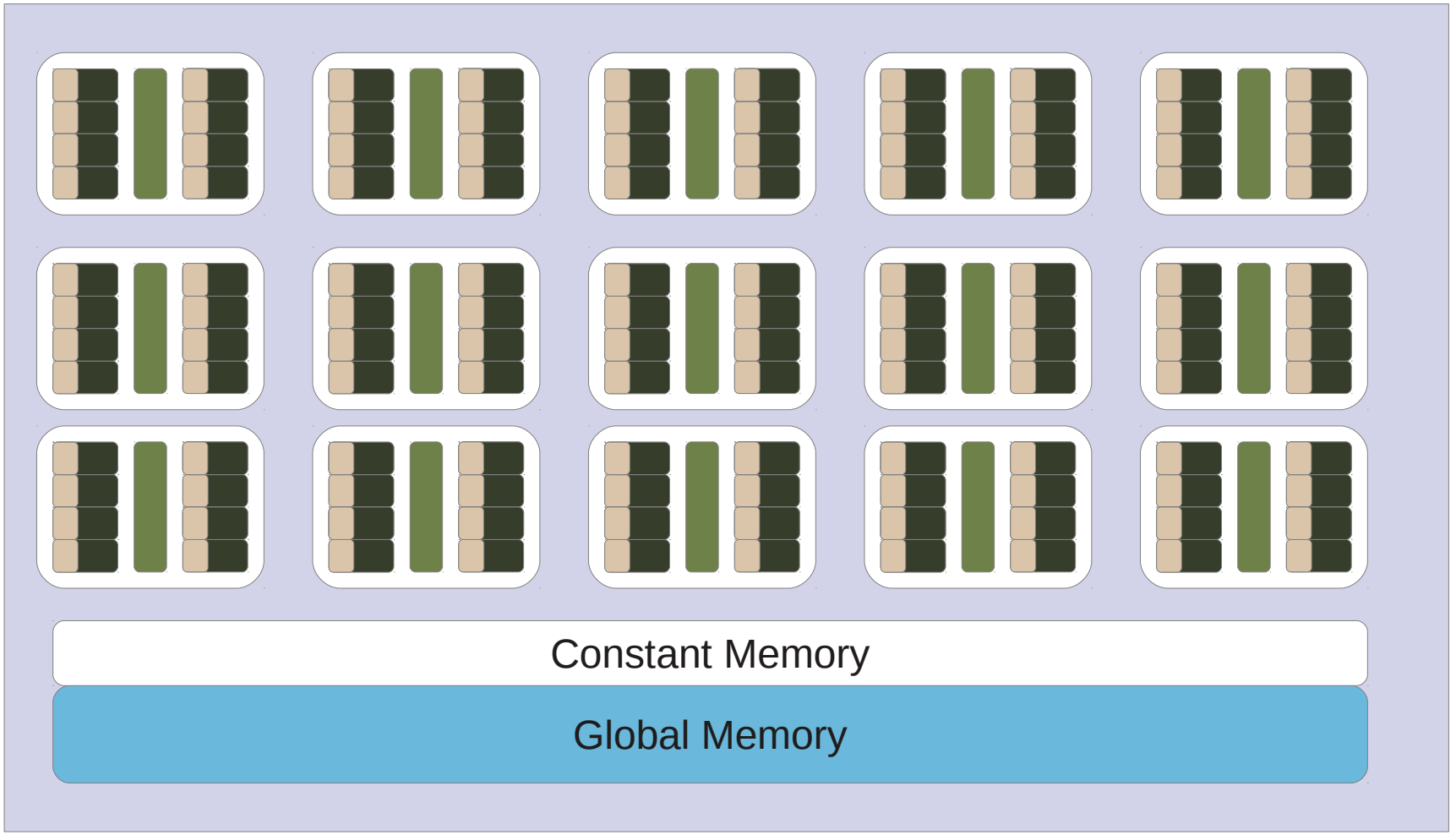
Private memory is accessible only by the processing element.

Local Memory

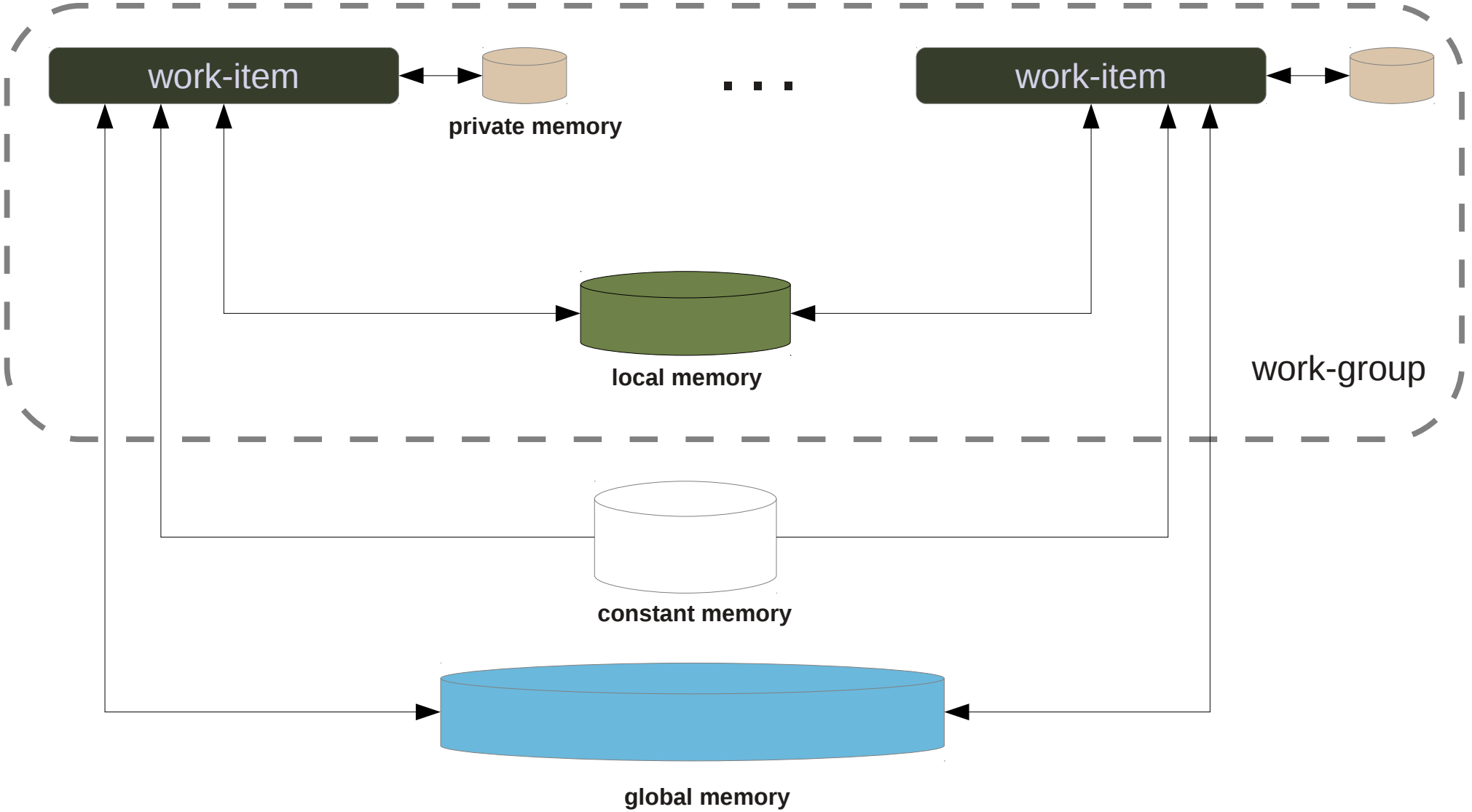


Local memory is shared by all PEs in a CU.

Inside the Device



OpenCL 1.2: Memory Model



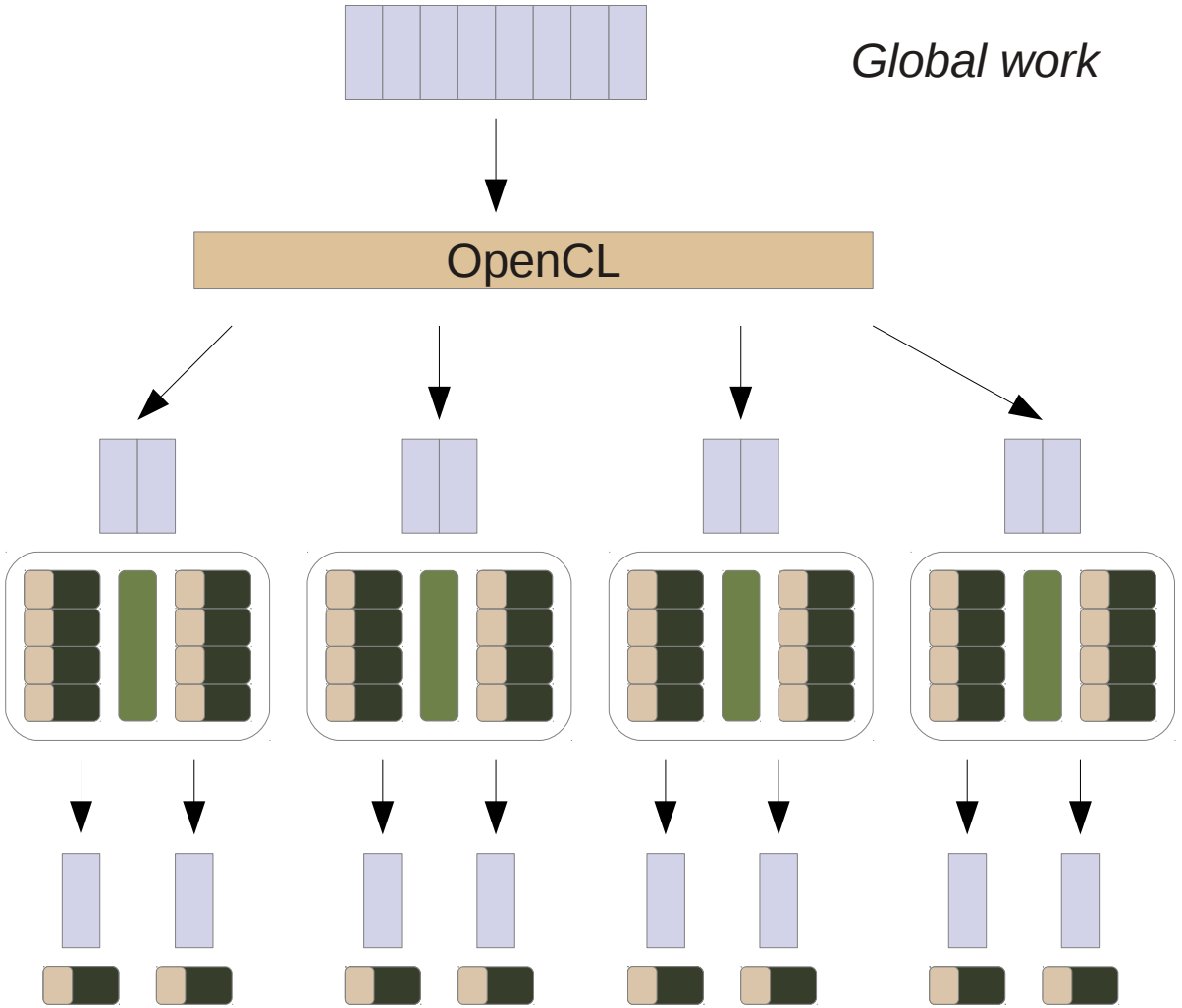
Execution Model: ND-Range Kernel

- You have some amount of data to process
  - Each “piece” of work has an address

    May have 1, 2, or 3 dimensions to the address
  - Addresses are unique in global space
- Data is processed in batches
  - This is called the *work-group*
  - Work-groups share local memory
- A single piece of data is called a *work-item*

ND-Range Kernel

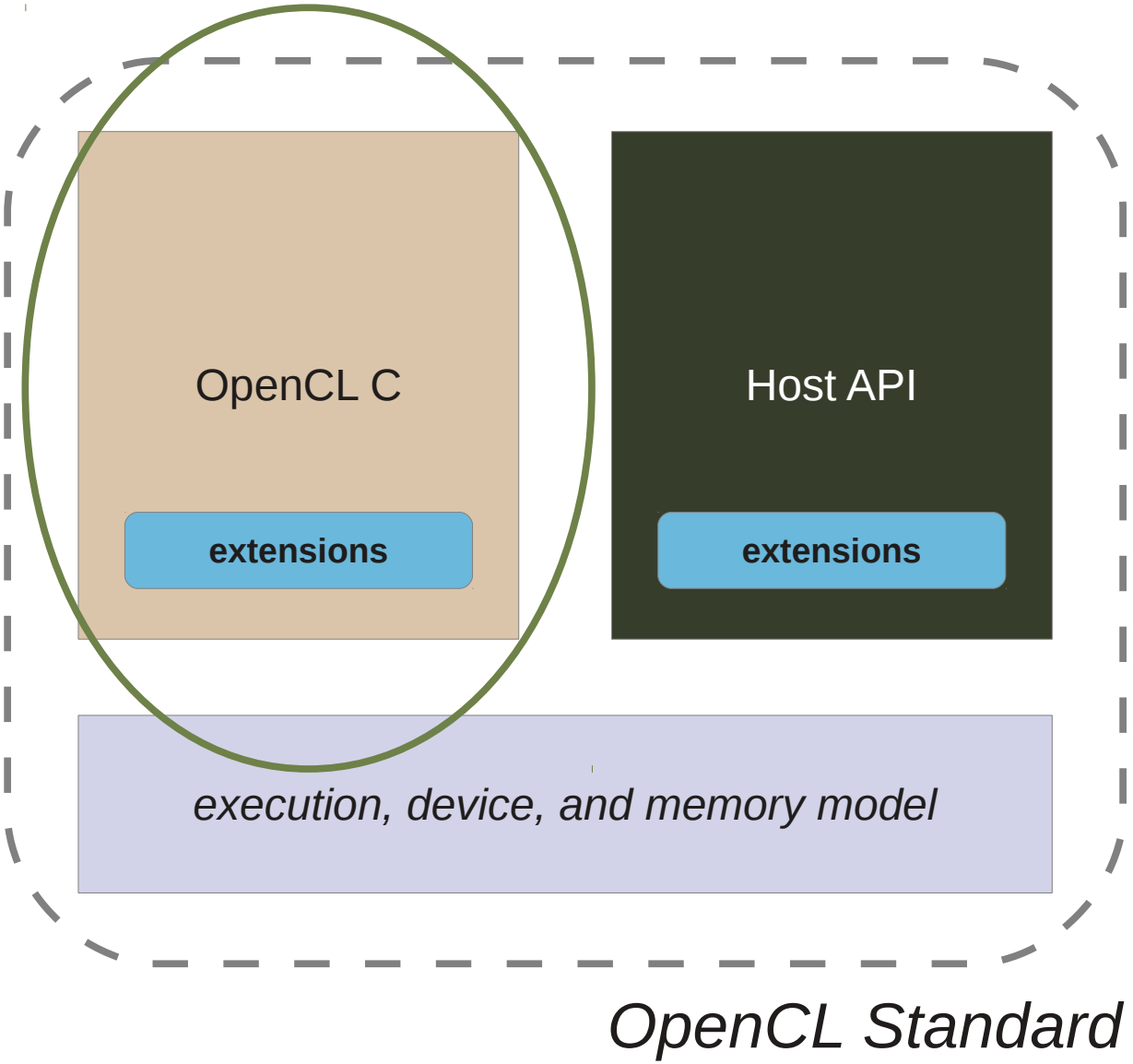
8 work-items, 1D execution, work-group size is 2.



End of Tour!

These are deep topics, we are only scratching the surface today.

OpenCL Components

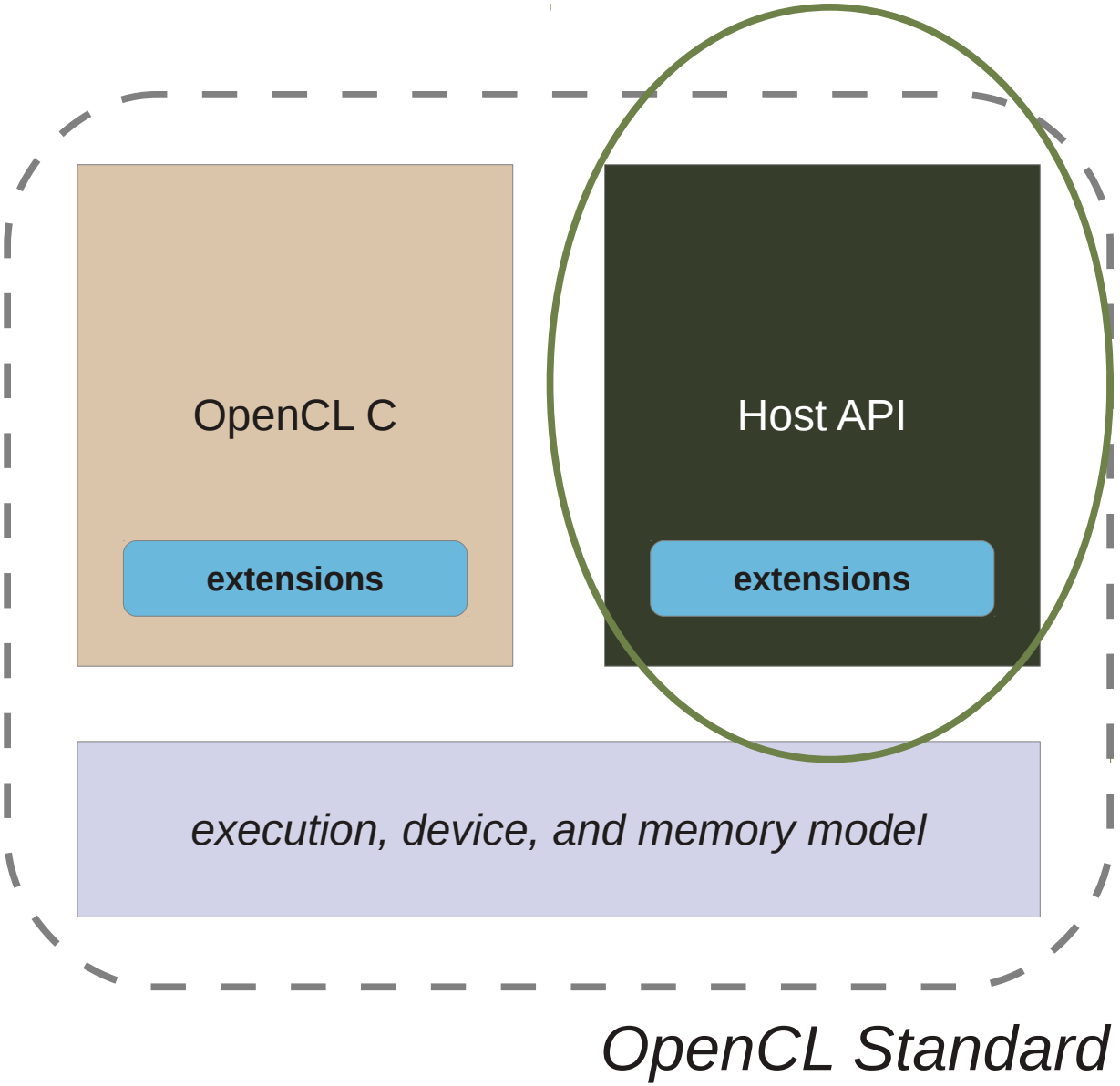


OpenCL C

- C99 with some modifications
  - global, constant, local, private memory regions
  - no function pointers

    No virtual functions!

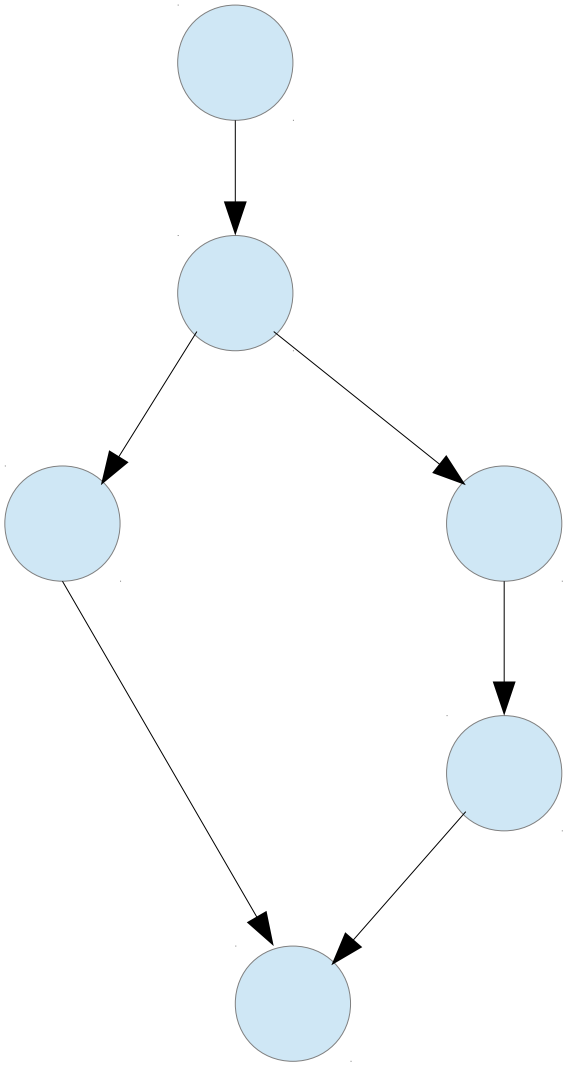
OpenCL Components



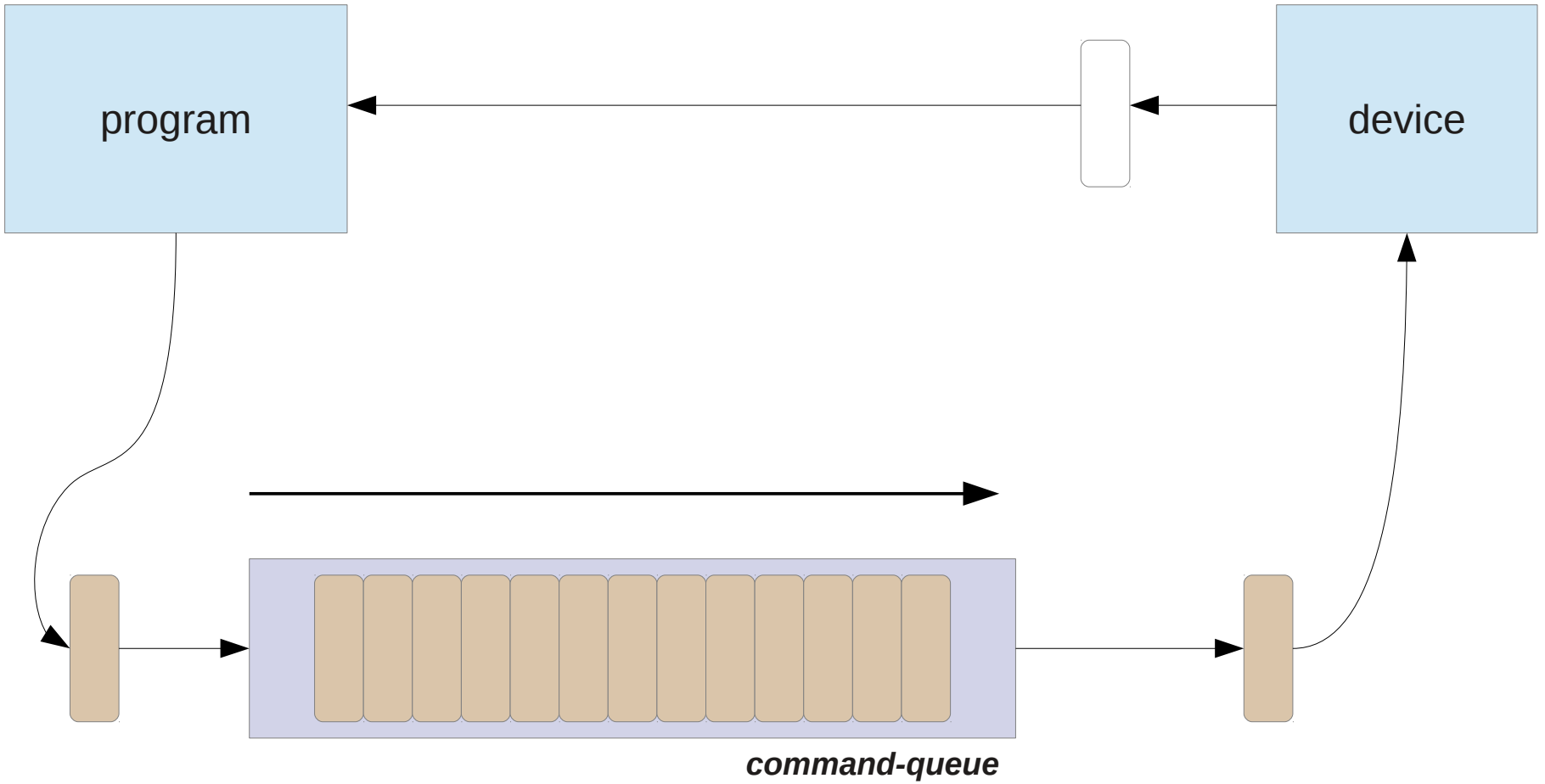
OpenCL Host API

- Platform (i.e. implementation of OpenCL)
- Device information
- Memory management
- Program compilation and loading
- Asynchronous commands
  - Memory transfer
  - Function execution
  - Memory map

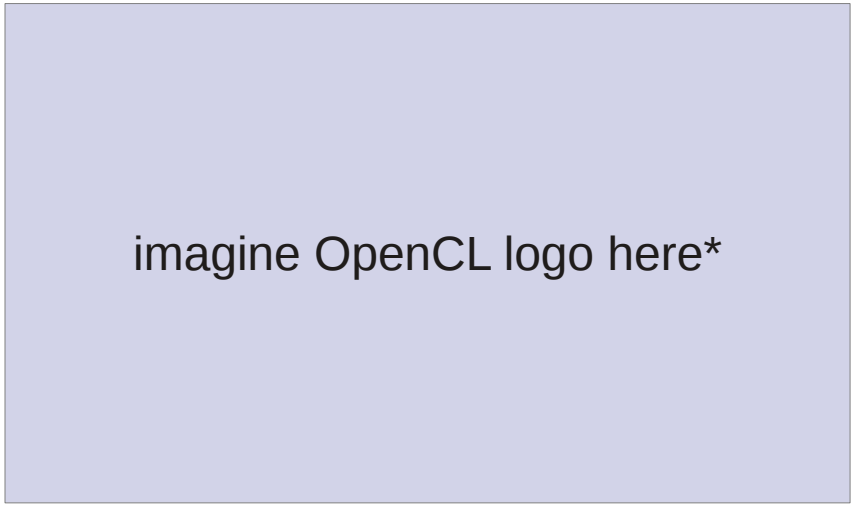
OpenCL Asynchronous Execution



OpenCL Command Dispatch



OpenCL 2.0



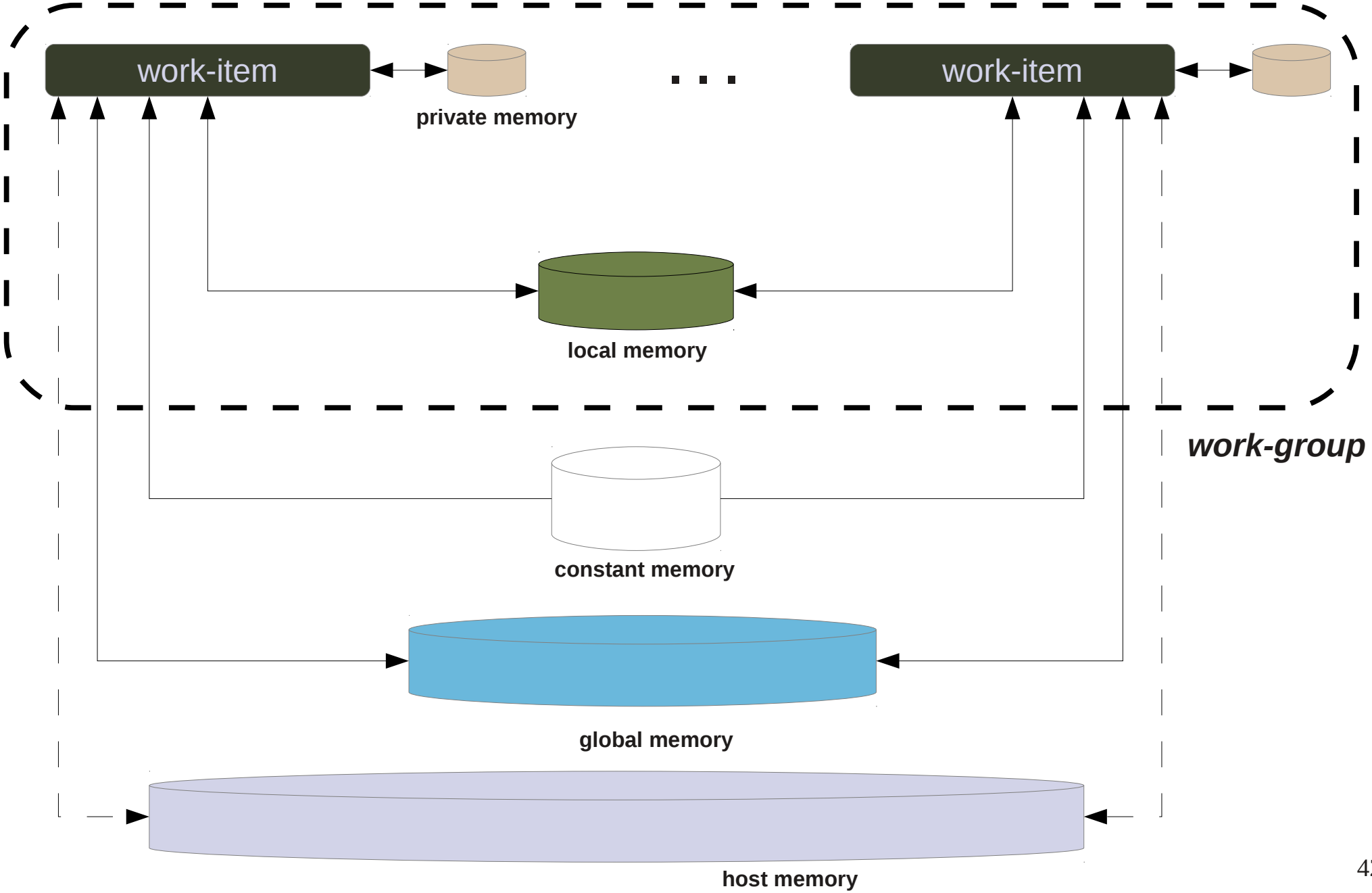
** Apple provides a horribly complicated legal framework to use the OpenCL logo, so let's use our imagination!*

OpenCL 2.0: Changes

- Support for shared virtual memory (SVM)
  - Allow devices to access memory on host
  - C99 atomics with memory consistency models
  - Devices virtual address space

    This means pointers are portable!
- Device can enqueue kernels itself
- Many more changes, but not in scope of talk

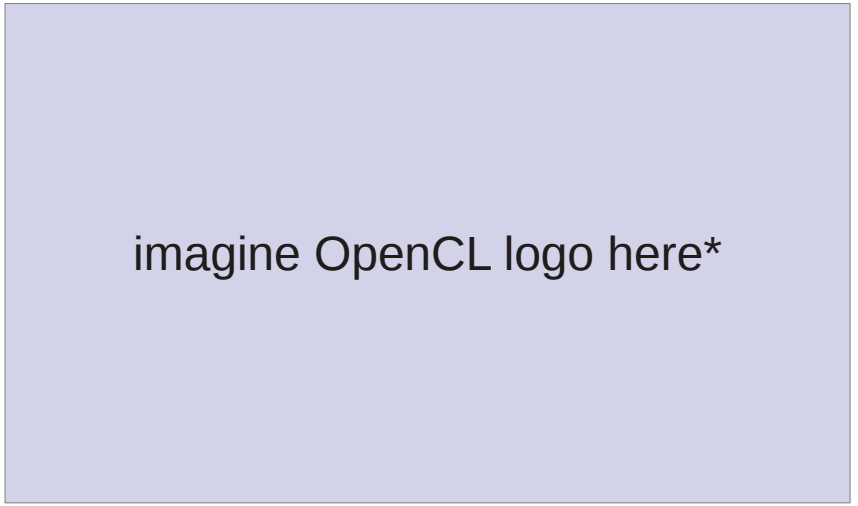
OpenCL 2.0: Memory Model



SPIR 1.2

- Standard Portable Intermediate Representation
  - LLVM IR
- Allows you to distribute instructions without source
  - i.e. you don't have to share the C code
- Allows you to write your own compiler
  - i.e. for languages other than OpenCL C

My OpenCL Feedback



** Apple provides a horribly complicated legal framework to use the OpenCL logo, so let's use our imagination!*

OpenCL 1.2: General Concerns

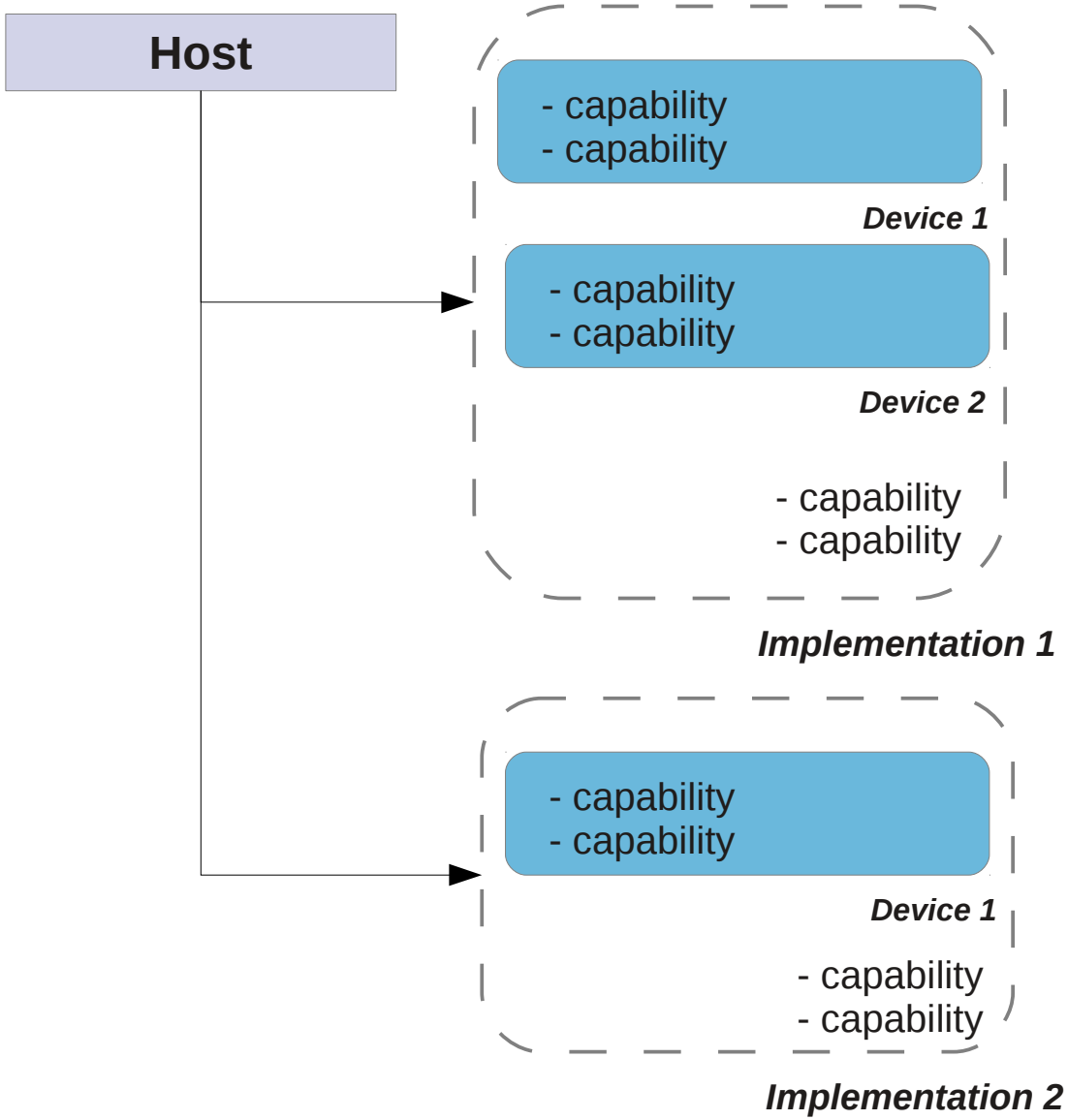
- Error model is not well-defined
  - All bets are off in the event of failure

    You might not even be the cause!
  - I have proposed an asynchronous error model
- Work-item termination
  - Allow work-items to trigger host exceptions
- OpenCL C type inspection
  - Required for middleware type-safety
- Various technical issues
  - See my blog for details

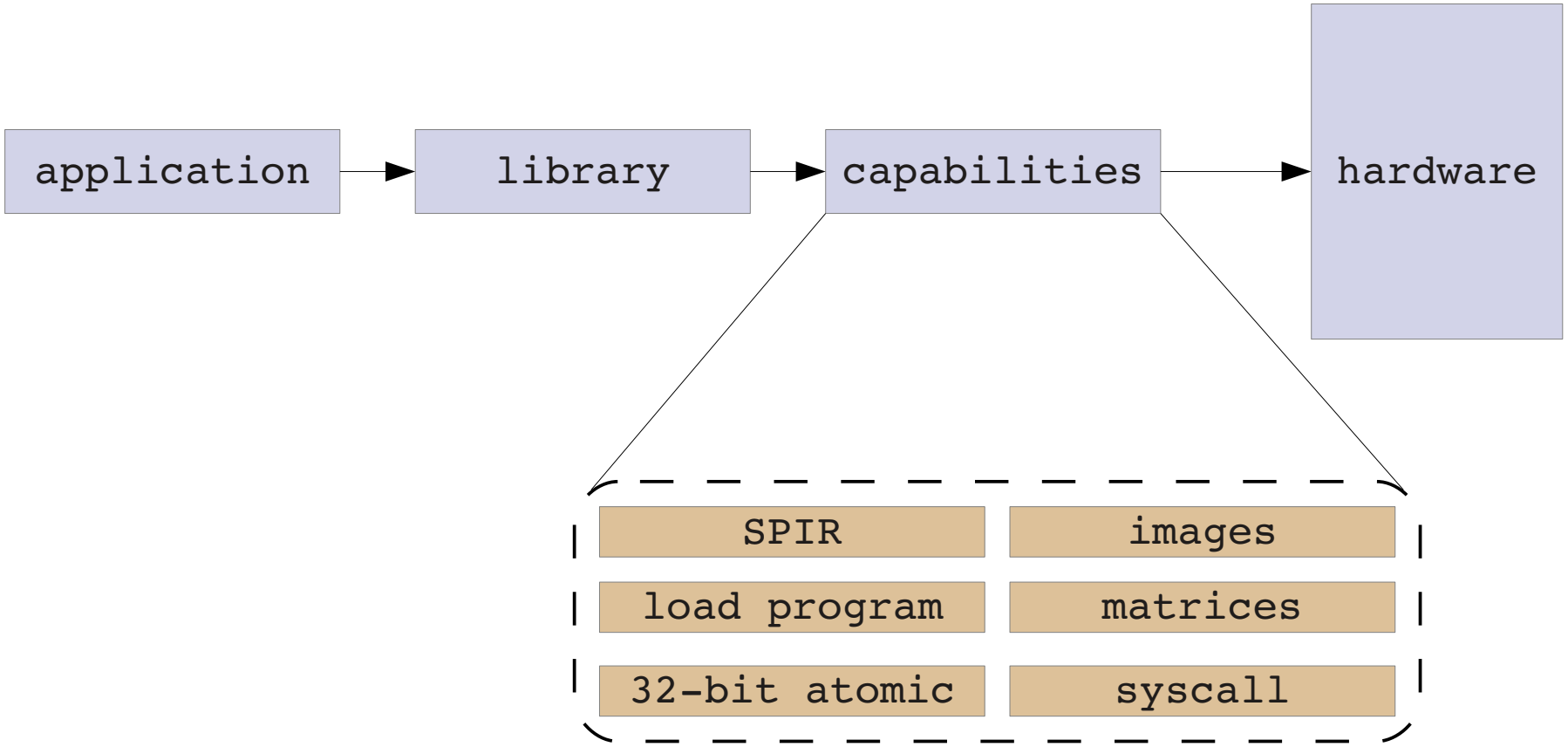
Common Theme

Separation of hardware and software concerns

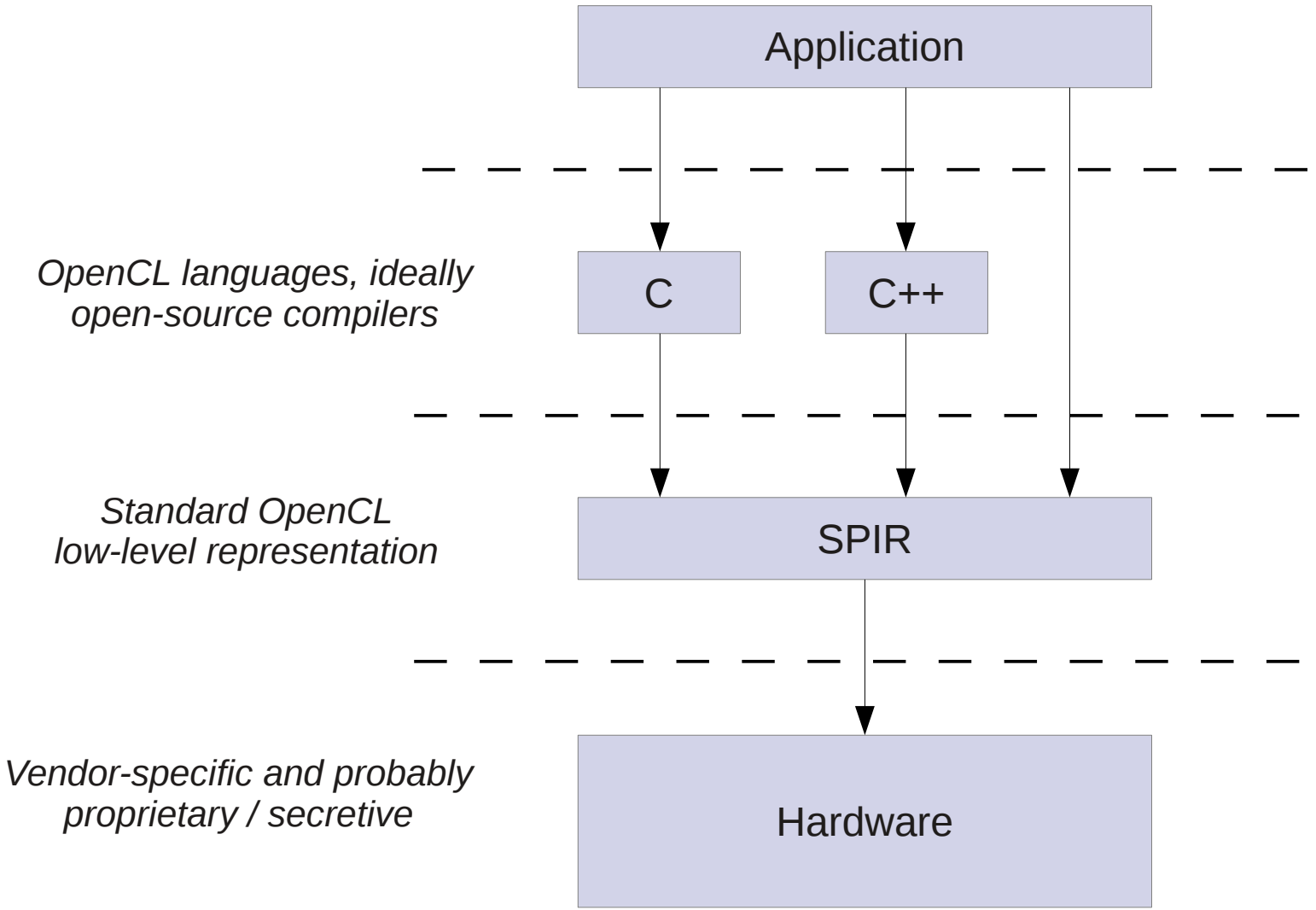
Capability Interface



OpenCL Application Architecture



SPIR Architecture



Why Change OpenCL Now?

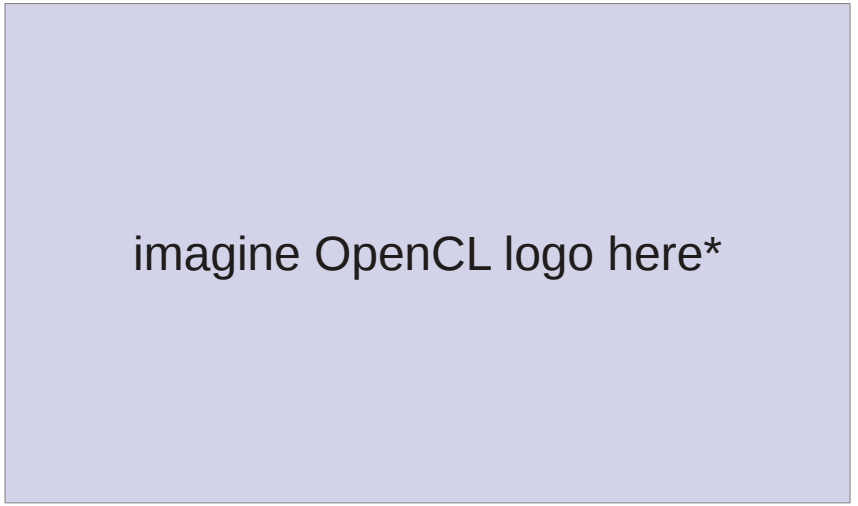
“Deal with a thing while it is still nothing;
keep a thing in order before disaster sets in.”

– Lao Tzu

My Standard Goals

- Make OpenCL future-proof
  - Software written today will run in future
- Reduce scope of OpenCL
  - It does too much!
  - It should only provide a software-hardware interface
  - Leave most details to software community
- Prepare for volatility
- Help develop software market

OpenCL Philosophy Revisited



** Apple provides a horribly complicated legal framework to use the OpenCL logo, so let's use our imagination!*

Revised Philosophy

- OpenCL is a low-level standard
- There is no “correct” set of abstractions
  - Devices can do different things
  - Compatibility layers can provide obsolescence protection
- Middleware provides abstractions
- OpenCL provides hardware as *services*
  - Libraries orchestrate devices

My Projects

Current Projects

- Capability interface (LGPL?)
  - Rather than waiting I'm doing this myself
  - Provide compatibility with OpenCL 1.0, 1.1, 1.2, 2.0

    I need help from the OpenCL standard, but I can demonstrate the value now
  - Delivery: one month
- C++11 OpenCL Library (dual-licensed GPLv3)
  - Delivery: end of year
  - Contact me if you want an early build
  - Contact me if you want to supply feature requests

Consulting

Consulting work funds my open source work!

http://blog.ajguillon.com

aj@ajguillon.com