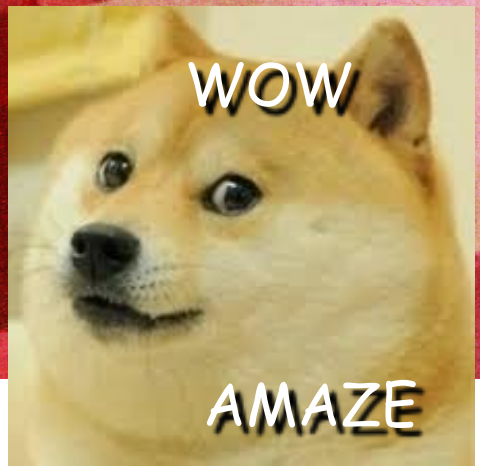


MUCH MODELS! MANY SIMPLICITY?

Raphael 'kena' Poss, University of Amsterdam

Putting Heterogeneous High-Performance Computing at the Fingertips of Domain Experts
Shonan Village Center, Japan, November 17-20, 2015

WHO'S THIS GUY ANYWAYS?

Trained with a HP library for **generic image processing** (Olena)
Created a HP library for **generic FSM processing** (Vaucanson)

Teaching: C / Unix, compiler construction

Wisdom acquired:

C++ with templates = HPC + functional programming without the name

Open standards essential for quality and against vendor lock-in

WHO'S THIS GUY ANYWAYS?

Software engineering with **enterprise RDBMS + embedded DSL**
Specialized in **HP object-oriented distributed data stores**

Wisdom acquired:

Locality is key

Data layout drives algorithm design, not the other way around

You can't (often) choose your layouts in real-world apps

People are really uncomfortable with too rapid changes

WHO'S THIS GUY ANYWAYS?

Hired to create a "automatic parallelizing C compiler"
for a **new dataflow/VN hybrid micro-arch arranged as many-core**;
Ended up writing its **codegen back-end and OS** instead

Teaching: Parallel programming, compiler construction

Wisdom acquired:

Hardware multi-threading absolutely essential to MP HPC
Dataflow scheduling cheaper than superscalar for the same benefits

However both are complicated to explain/teach/model/adopt

Programmers really do not understand caching

WHO'S THIS GUY ANYWAYS?

Assigned to work on the **unification of run-time back-ends** for a HPC **streaming dataflow coordination language** (Hi Clemens!)

Teaching: computer architecture

Wisdom acquired:

No coordination possible without operational semantics in PL
("pure FP for HPC is like trying to run with one leg")

Programmers do care about operational semantics
But: only a little!

Programmers do not understand implicit caches and explicit memory

WHO'S THIS GUY ANYWAYS?

2001-2003 | 2006-2008 | 2008-2011 | 2011-2013 | 2014-present

Own research: co-design of **future general-purpose multi-cores** with their software stack but also **abstract models**

Paid for: new processor for ESA, new OS for ASML, teaching materials

Teaching: computer architecture, operating systems

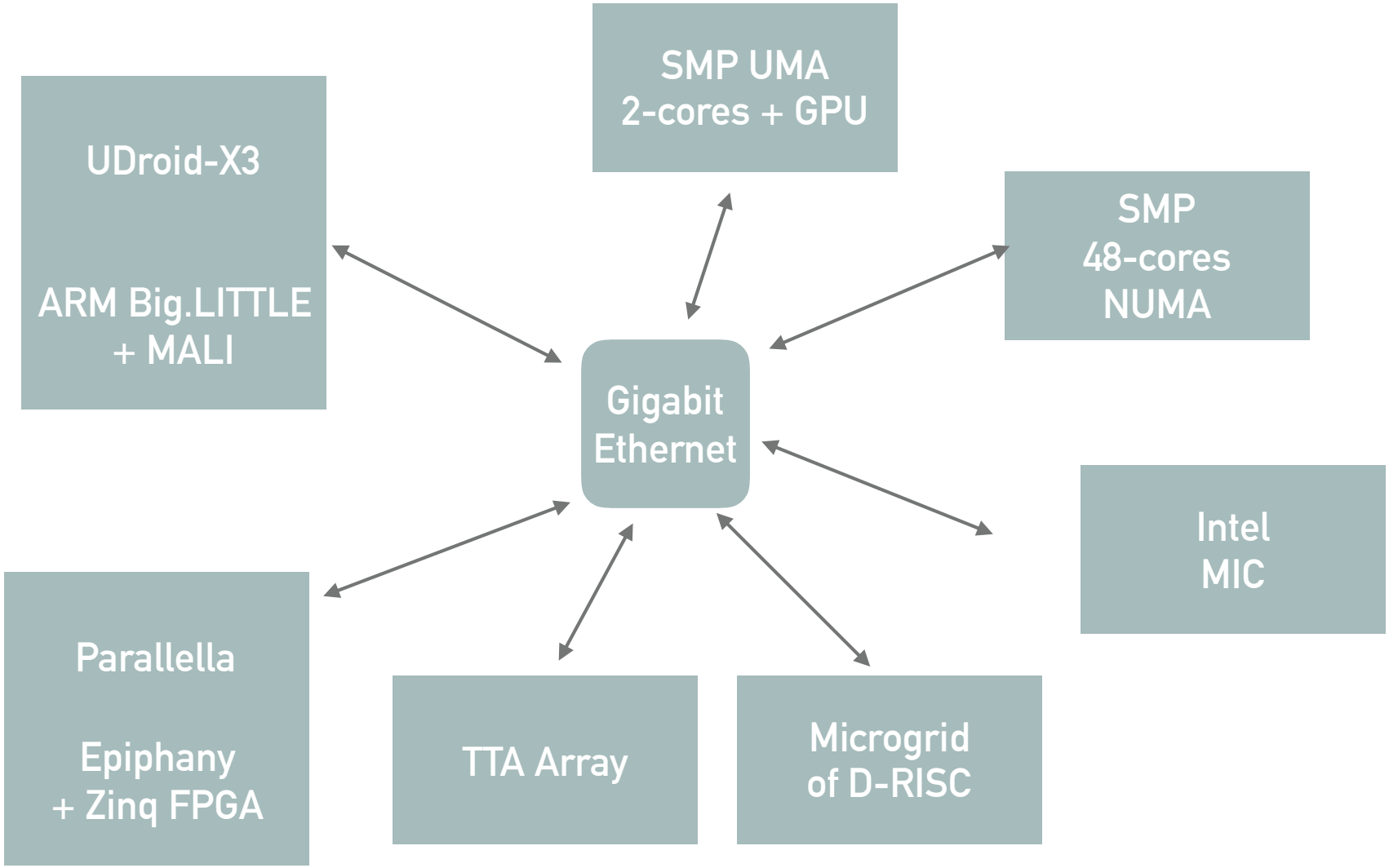
Wisdom acquired:

Unix is there to stay, and there are actually good scientific reasons

Programmers don't get Von Neumann, but they get BSP, Actors and PNs
Programmers don't get caching and memory, but they get locality

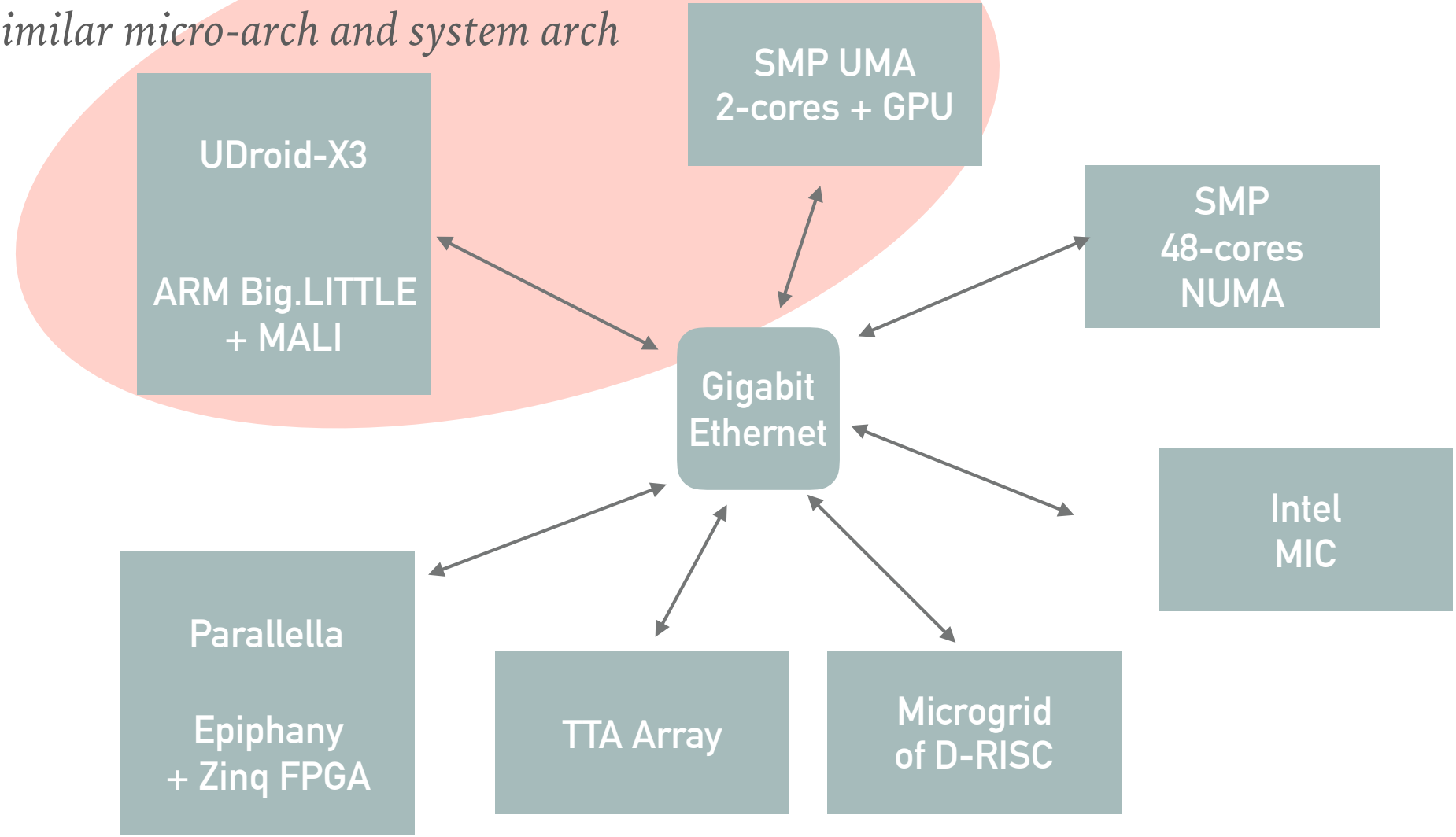
First 2 PL learned condition the entire intellectual career of a programmer

WHAT HETEROGENEITY MEANS TO ME



WHAT HETEROGENEITY MEANS TO ME

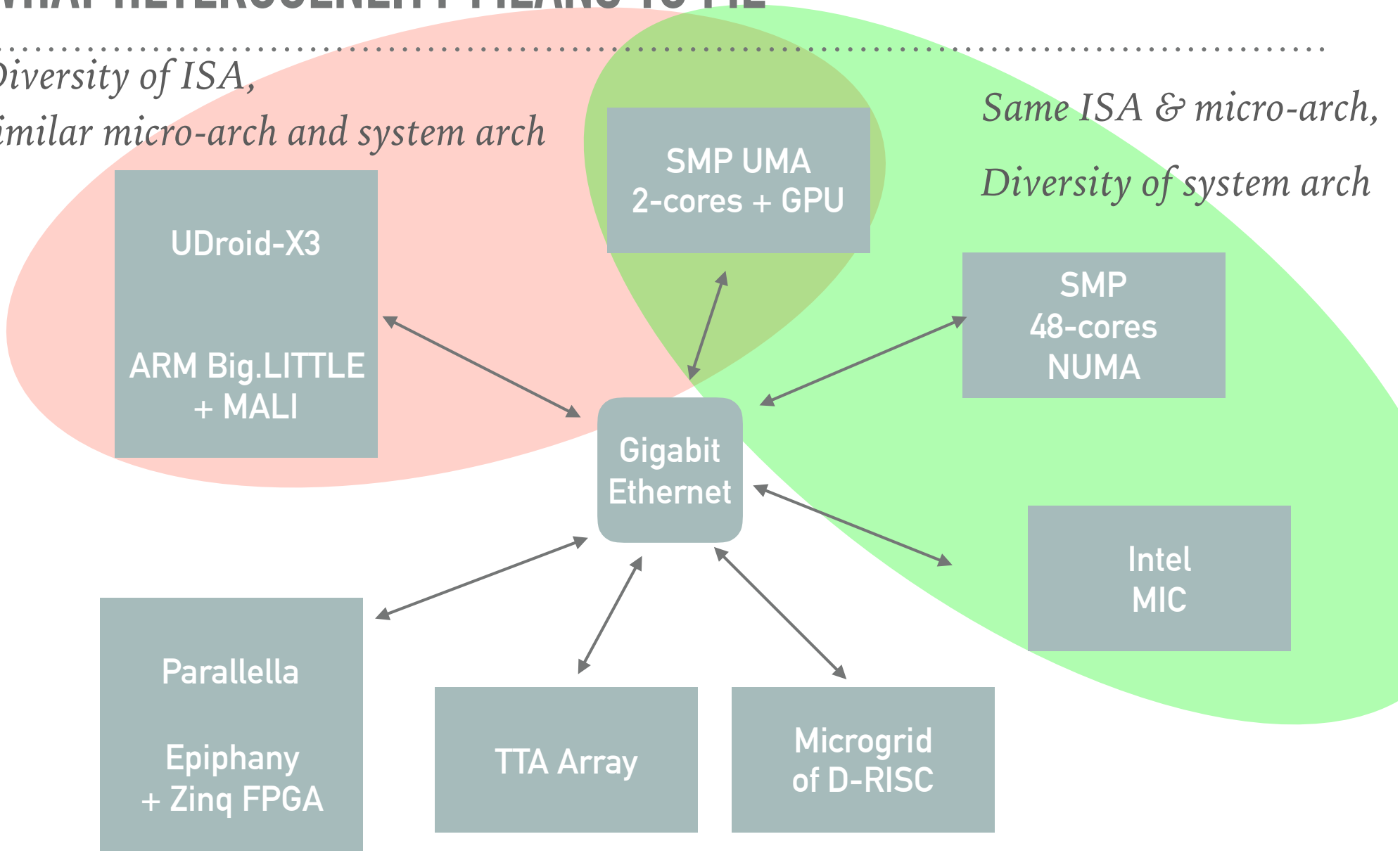
Diversity of ISA,
similar micro-arch and system arch



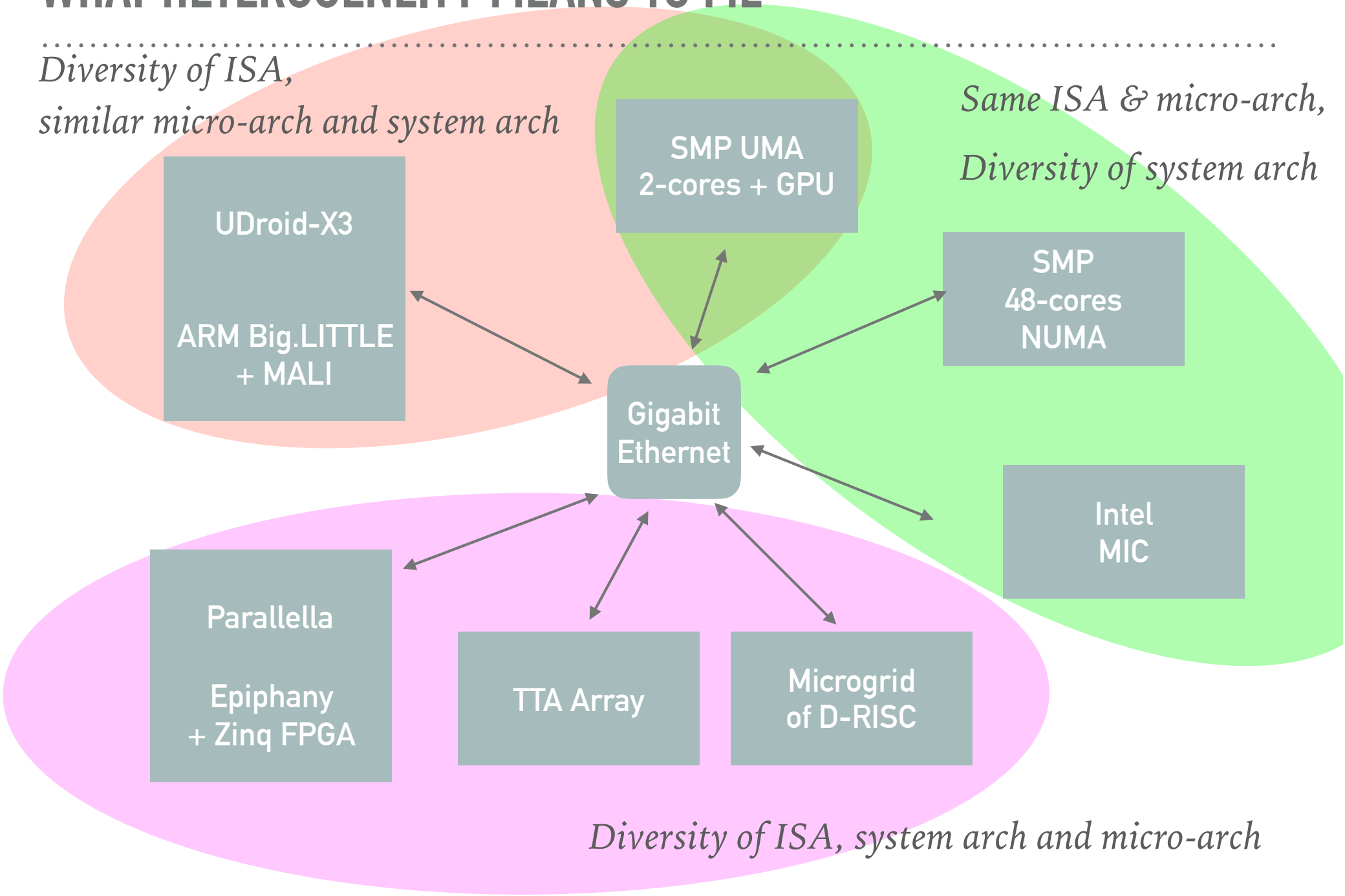
WHAT HETEROGENEITY MEANS TO ME

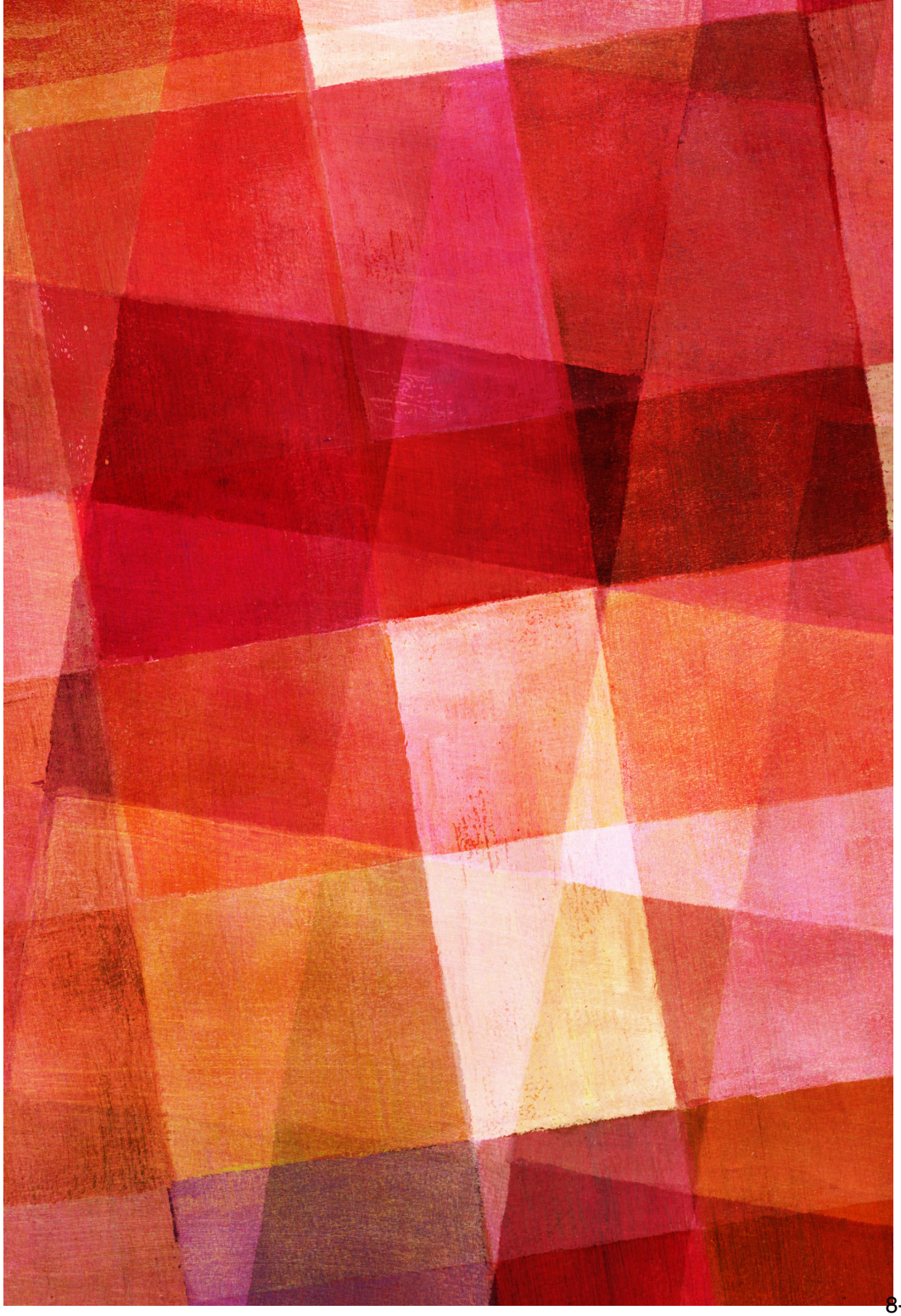
Diversity of ISA,
similar micro-arch and system arch

Same ISA & micro-arch,
Diversity of system arch

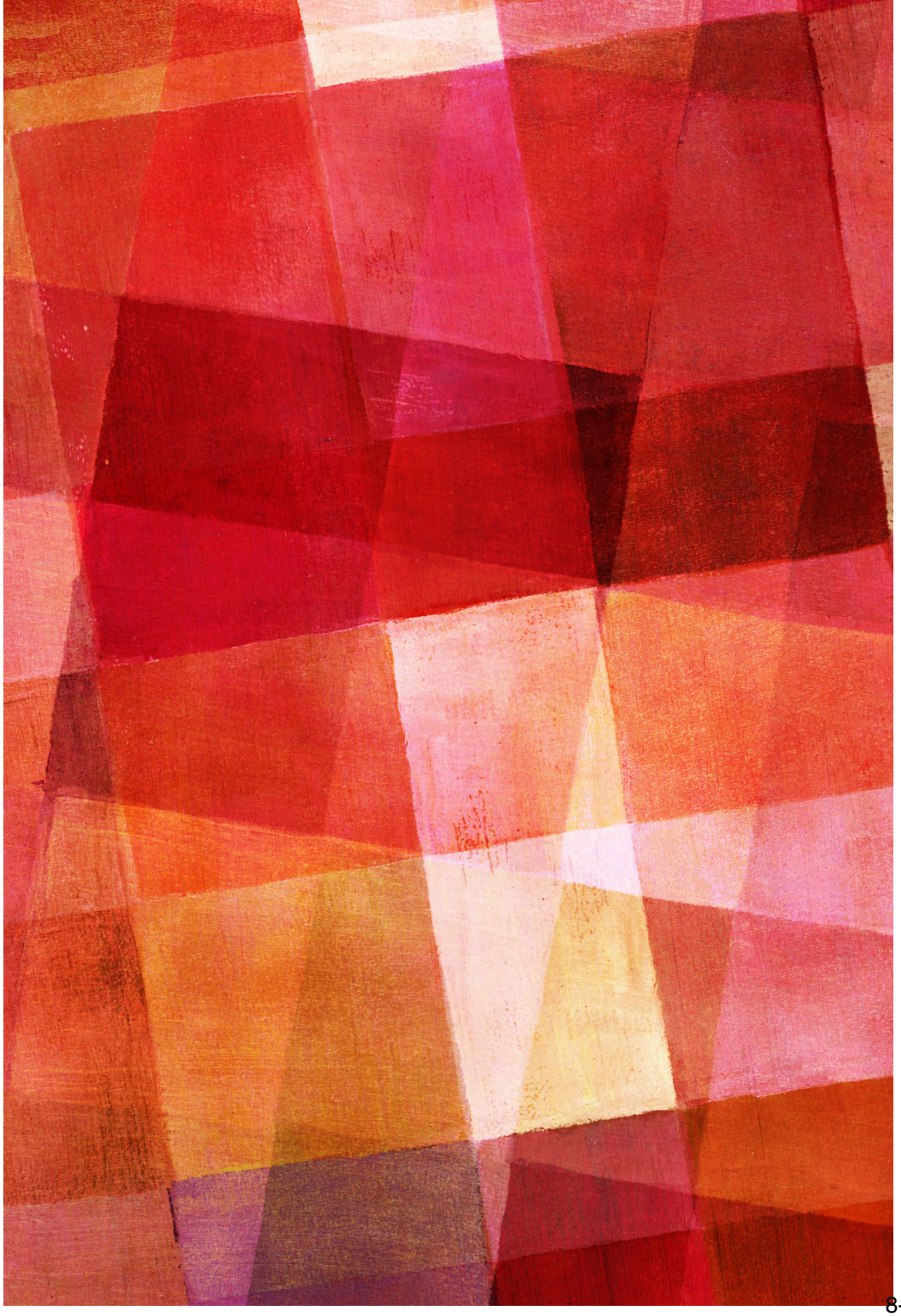


WHAT HETEROGENEITY MEANS TO ME



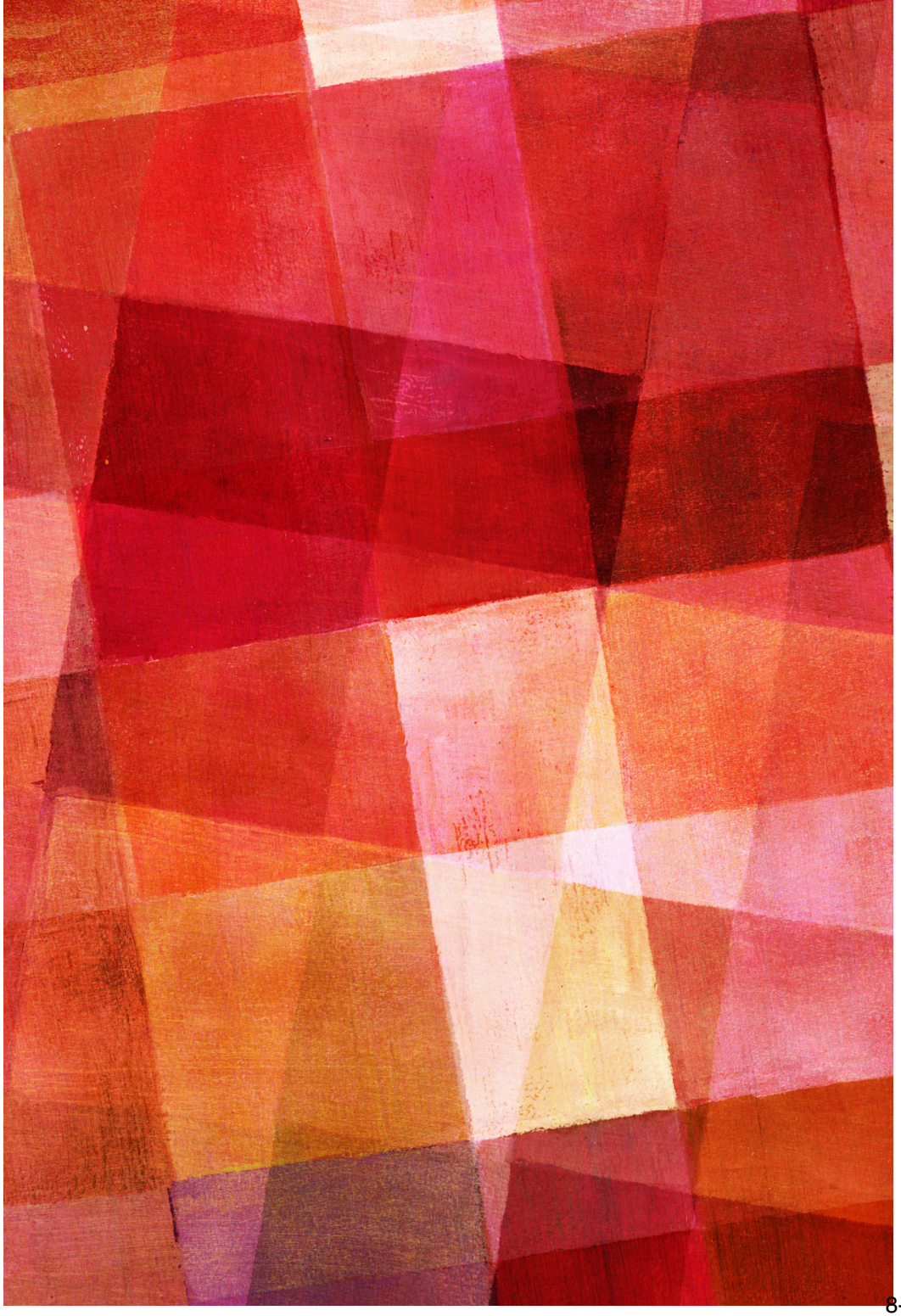


RESEARCH QUESTIONS



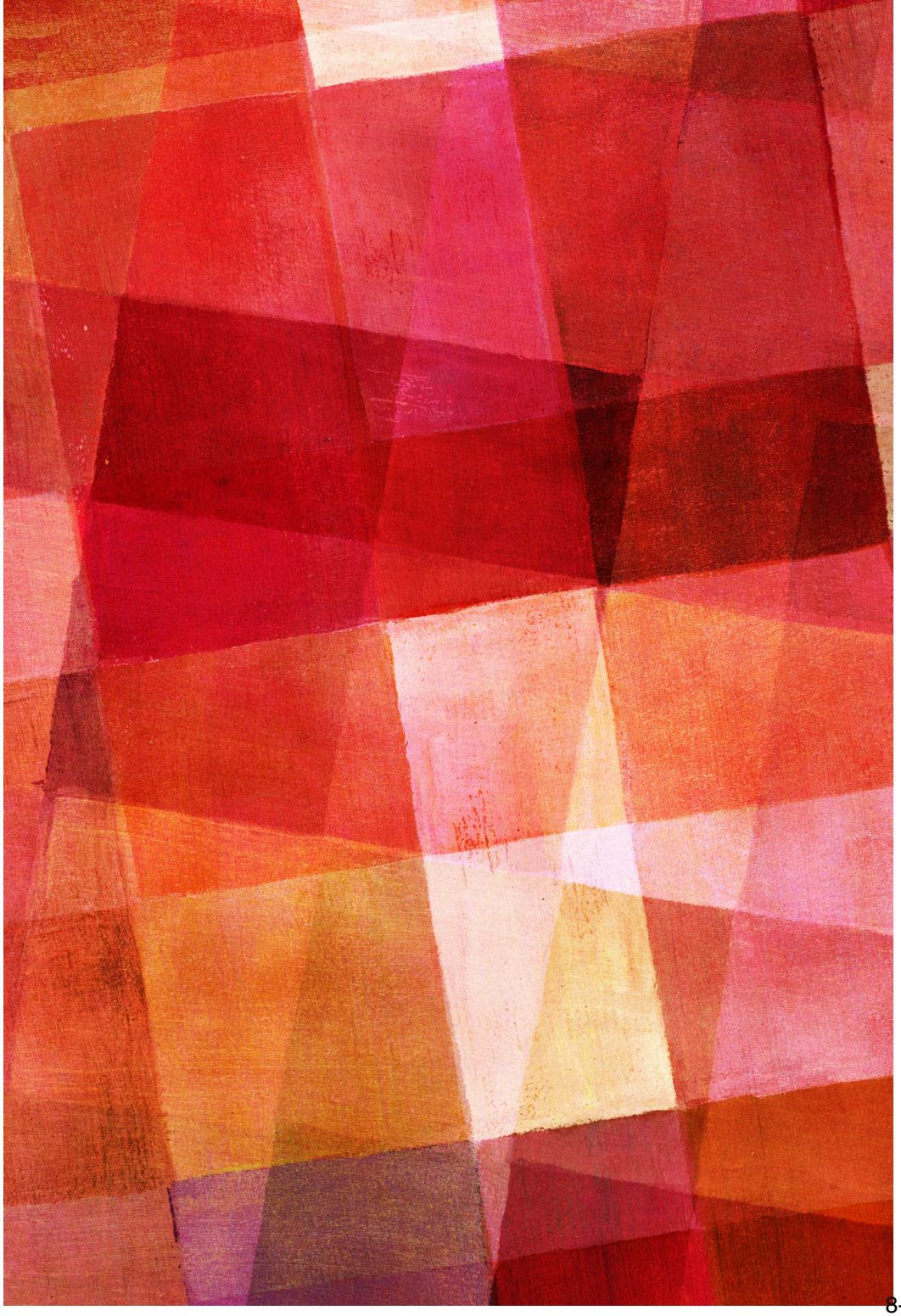
RESEARCH QUESTIONS

- How to explain
  complex parallel systems
  to 1st term undergrads
  so they can successfully write
  efficient & scalable code
  by their 2nd term?



RESEARCH QUESTIONS

- **How to explain complex parallel systems to 1st term undergrads so they can successfully write efficient & scalable code by their 2nd term?**
- What would Unix look like if designed for 21st century computers?



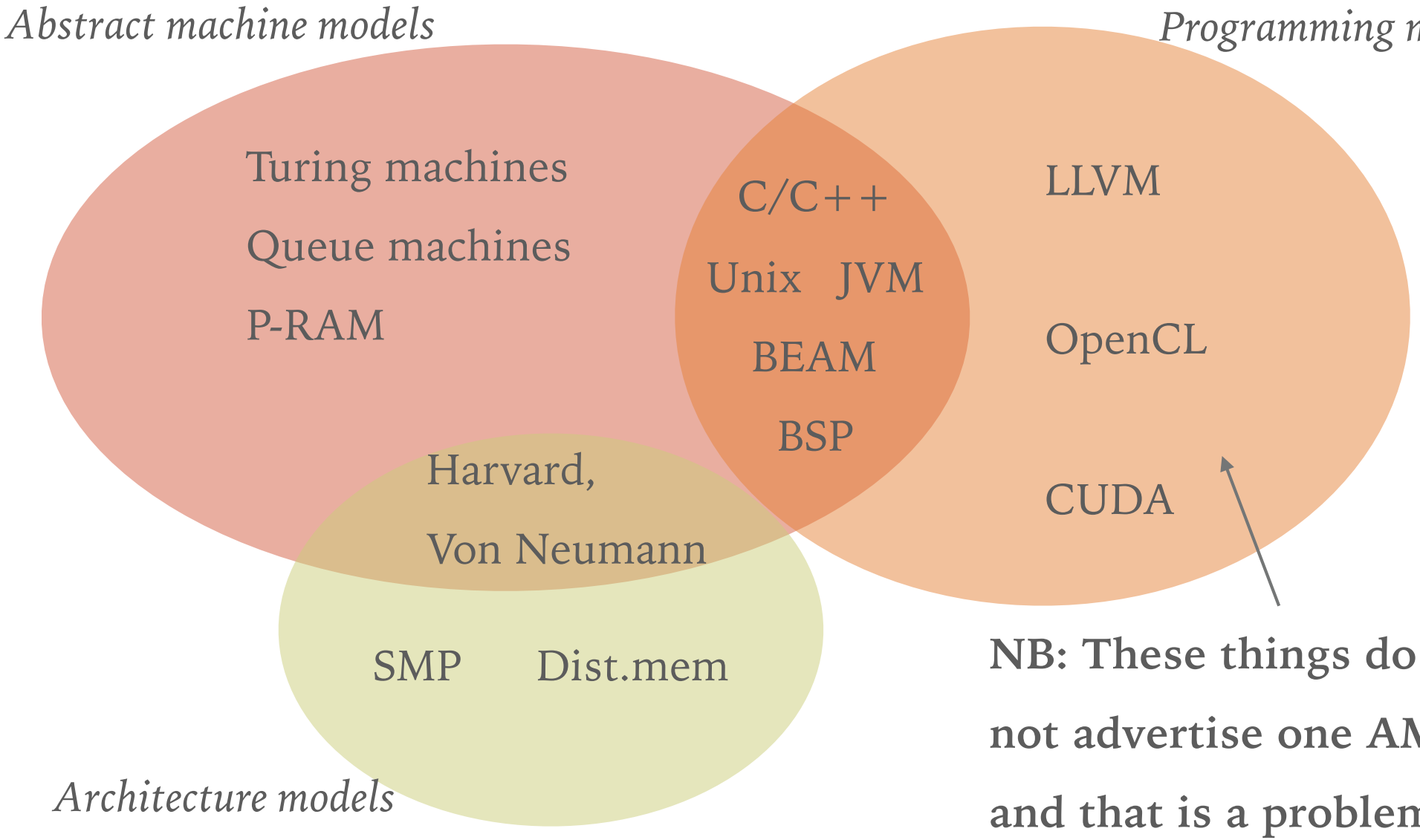
RESEARCH QUESTIONS

- **How to explain complex parallel systems to 1st term undergrads so they can successfully write efficient & scalable code by their 2nd term?**
- What would Unix look like if designed for 21st century computers?
- What should be the BASIC and C of our new computers?

WHAT'S AN AMM (ABSTRACT MACHINE MODEL)?

- *An abstraction of a class of computer systems sufficient to reason about its behavior while executing a class of algorithms*
- Example "pure AMMs" without a specific PL or code IR: Turing, Von Neumann, Harvard, P-RAM, Dataflow, BSP, Unix
- All modern PL AMMs are simulated upon a platform, using assumptions about the platform defined in the platform's AMM (usually VN/P-RAM)

TERM RELATIONSHIPS



NB: These things do not advertise one AMM, and that is a problem.

"Computing models" = Union of all three

PLATFORM ABSTRACT MACHINE MODELS, 1950–2015

PLATFORM ABSTRACT MACHINE MODELS, 1950–2015

- **Von Neumann / Harvard / P-RAM (threads, etc.)**
  - Building blocks: N active sequential processors + 1 RAM
  - Perf. predictive power only when programmed via BSP
  - **Performance scalability OR software composability, choose only 1!**

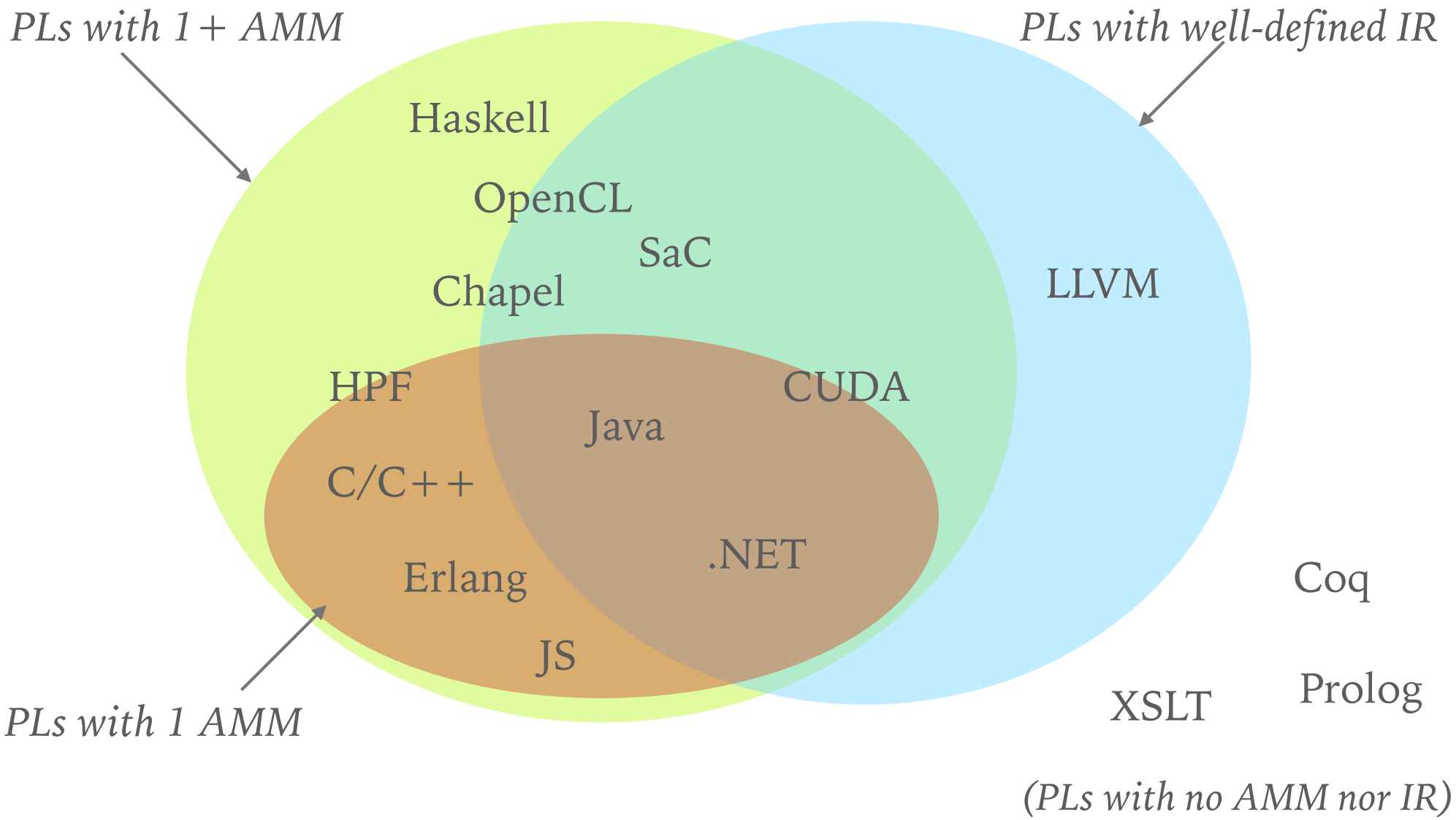
PLATFORM ABSTRACT MACHINE MODELS, 1950–2015

- **Von Neumann / Harvard / P-RAM (threads, etc.)**
  - Building blocks: N active sequential processors + 1 RAM
  - Perf. predictive power only when programmed via BSP
  - **Performance scalability OR software composability, choose only 1!**
- **Dataflow-like (incl. most RTS for lazy FP evaluation)**
  - Building blocks: N on-demand, "passive" functional units + 1 tag soup
  - Perf. predictive power only for small tag stores and static graphs
  - **Terrible at expressing priorities in mixes of data / control tasks**
    Poor at controlling locality

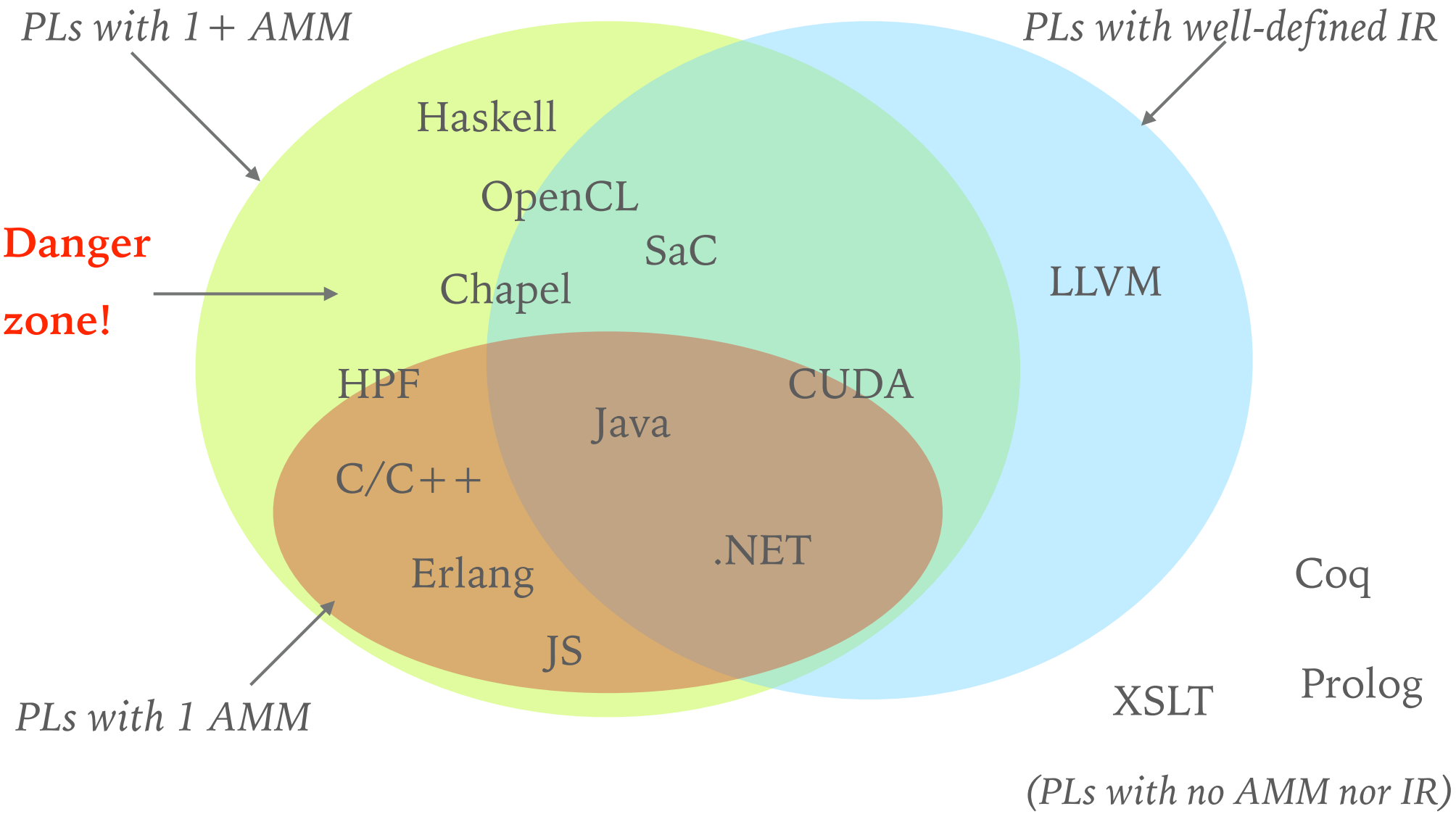
PLATFORM ABSTRACT MACHINE MODELS, 1950–2015

- **Von Neumann / Harvard / P-RAM (threads, etc.)**
  - Building blocks: N active sequential processors + 1 RAM
  - Perf. predictive power only when programmed via BSP
  - **Performance scalability OR software composability, choose only 1!**
- **Dataflow-like (incl. most RTS for lazy FP evaluation)**
  - Building blocks: N on-demand, "passive" functional units + 1 tag soup
  - Perf. predictive power only for small tag stores and static graphs
  - **Terrible at expressing priorities in mixes of data / control tasks**
    Poor at controlling locality
- **Especially missing: models for heterogeneous platforms,**
  ability to predict/control extra-functional behavior other than time

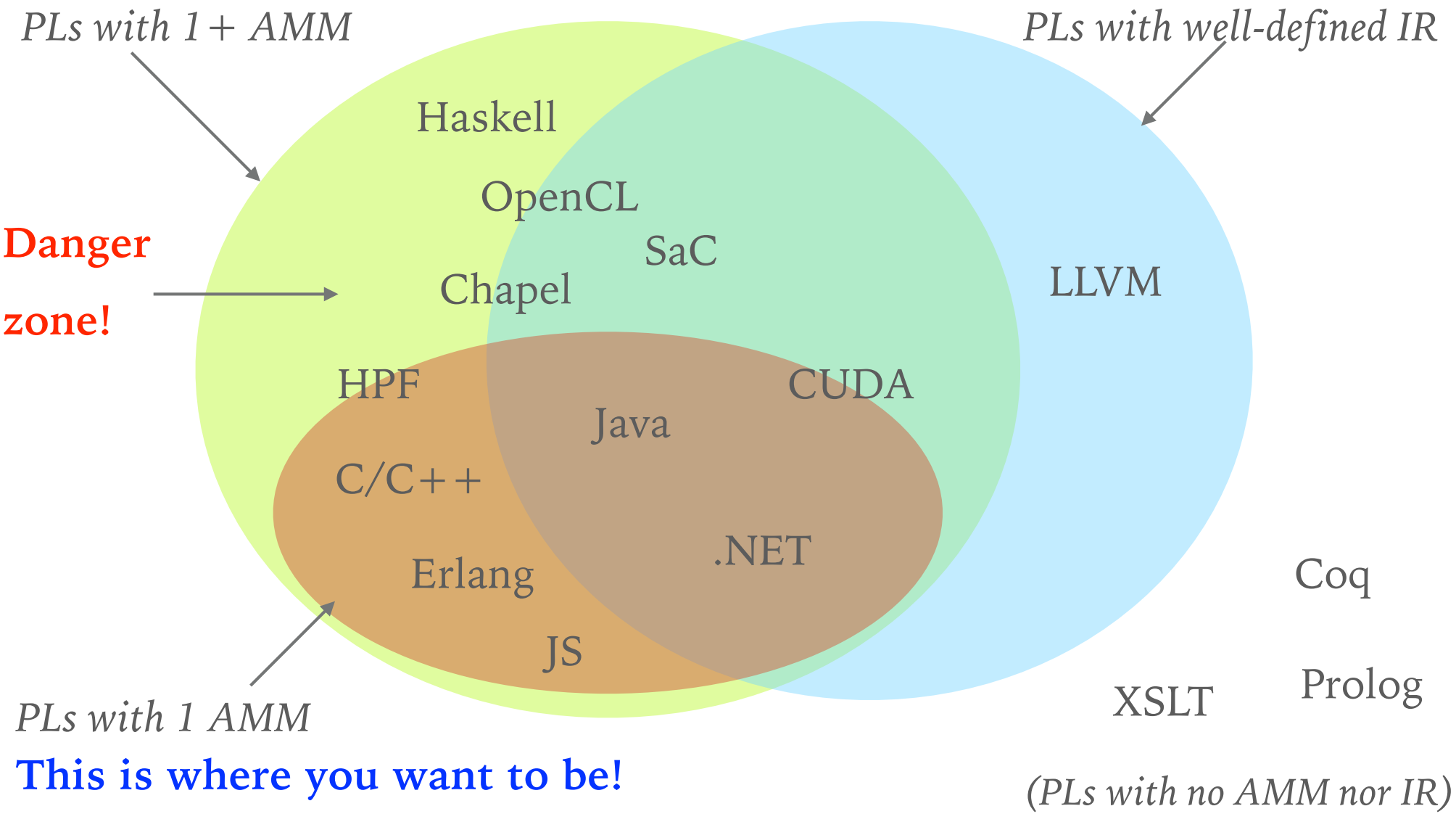
PL. VS. IR VS. AMM



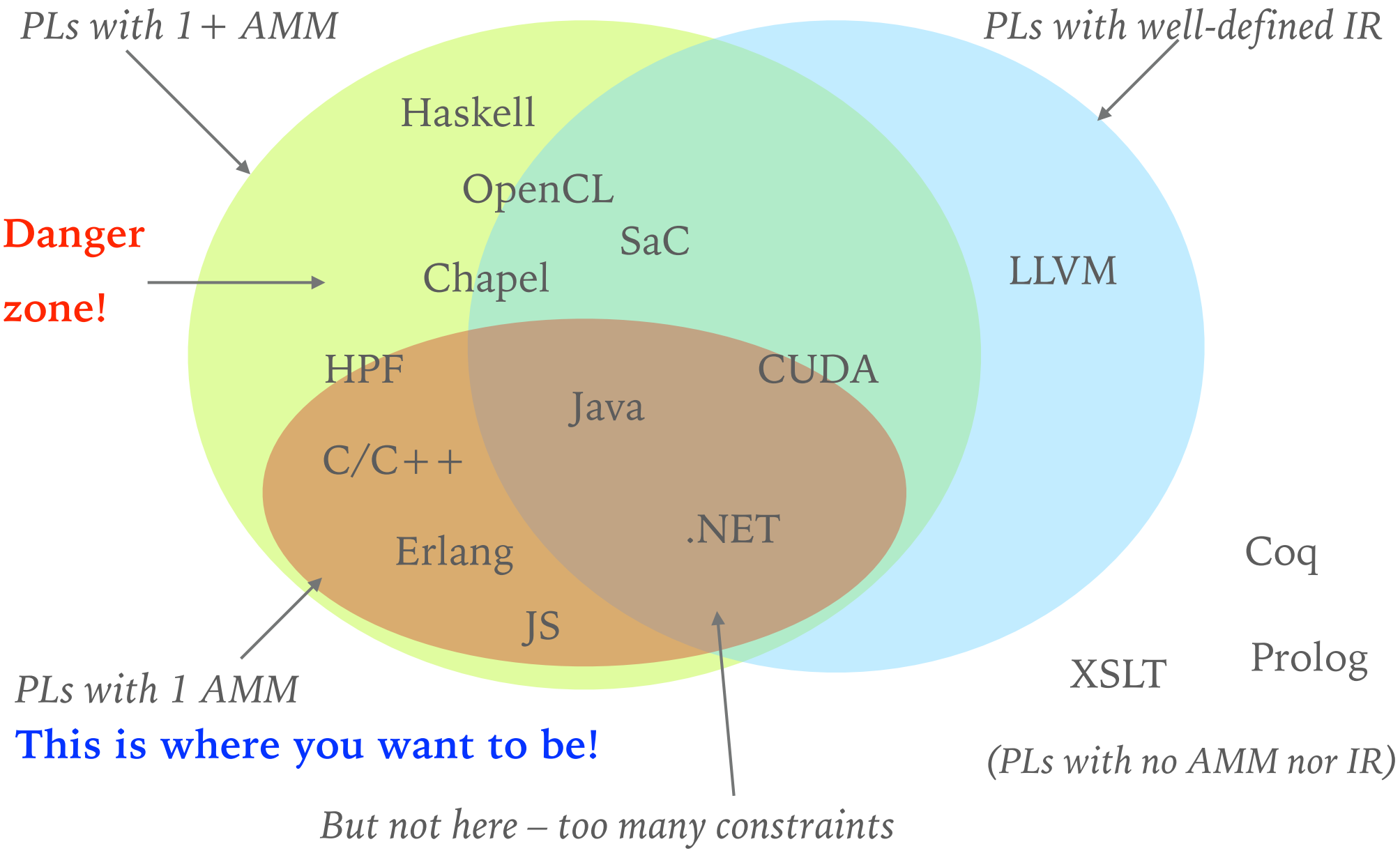
PL. VS. IR VS. AMM



PL. VS. IR VS. AMM

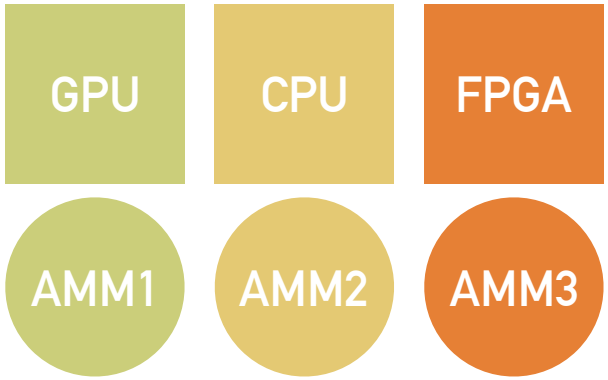


PL. VS. IR VS. AMM



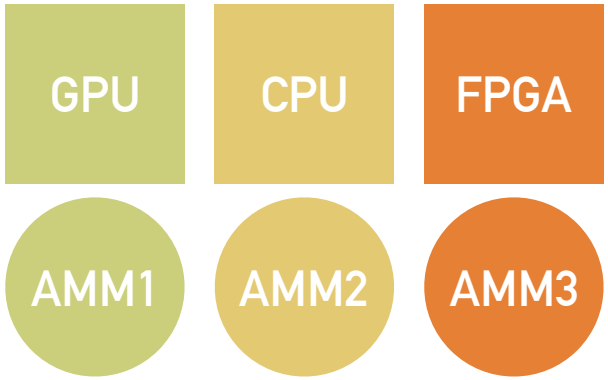
WISHFUL THINKING VS. REALITY

What we think we have:



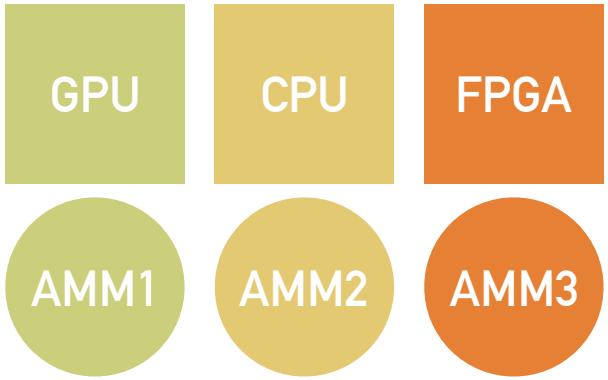
WISHFUL THINKING VS. REALITY

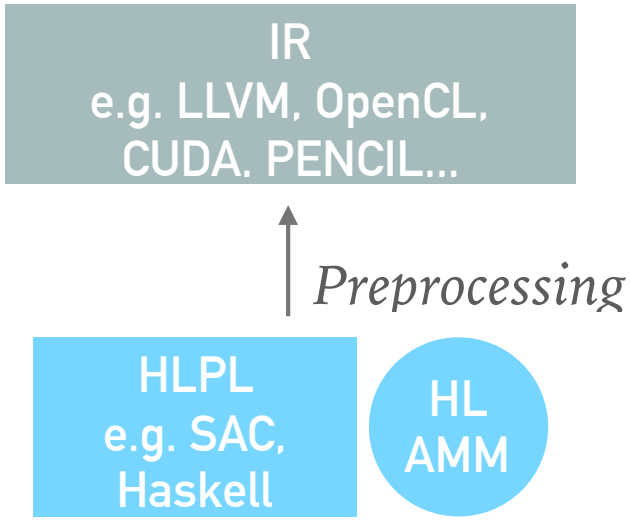
What we think we have:



WISHFUL THINKING VS. REALITY

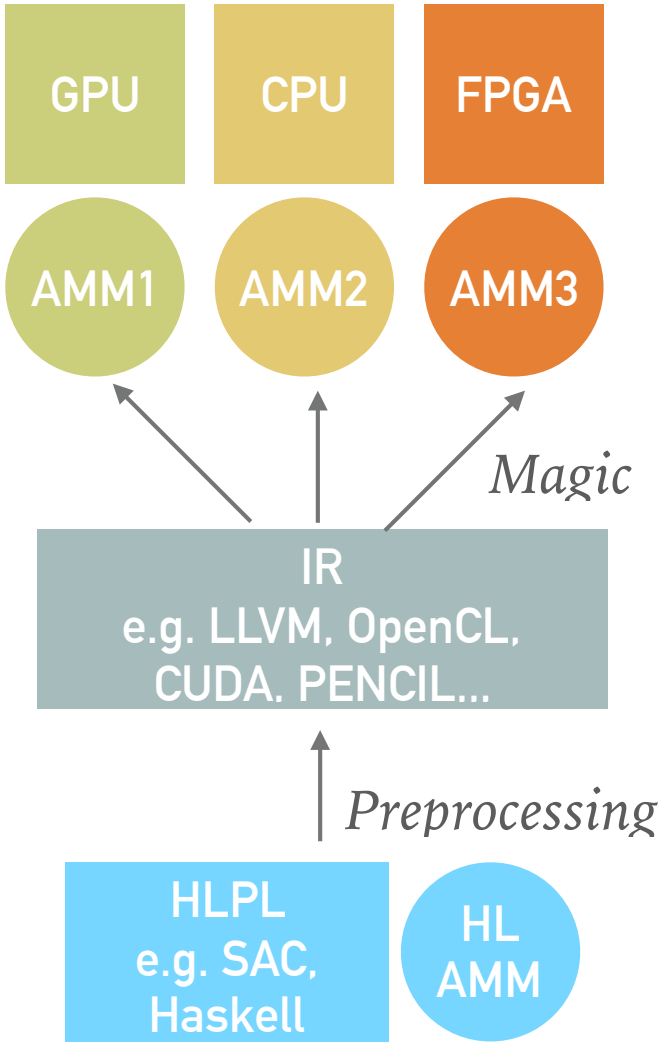
What we think we have:





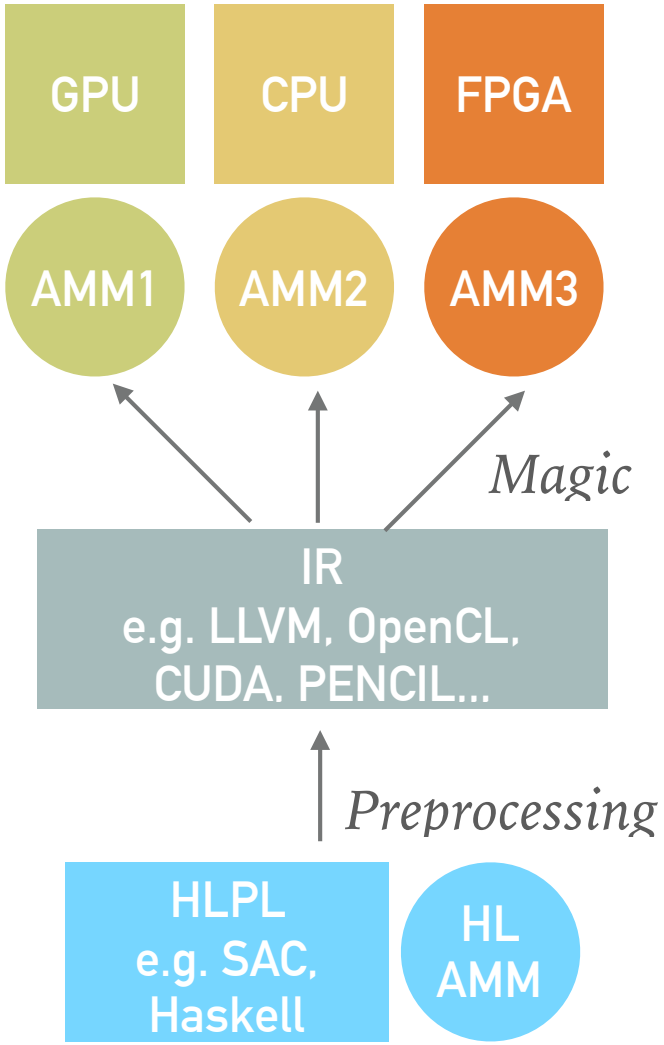
WISHFUL THINKING VS. REALITY

What we think we have:

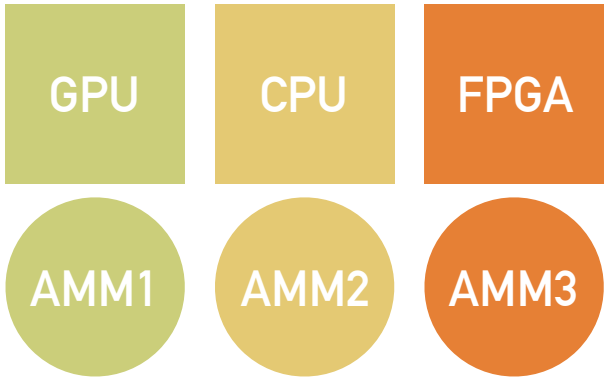


WISHFUL THINKING VS. REALITY

What we think we have:

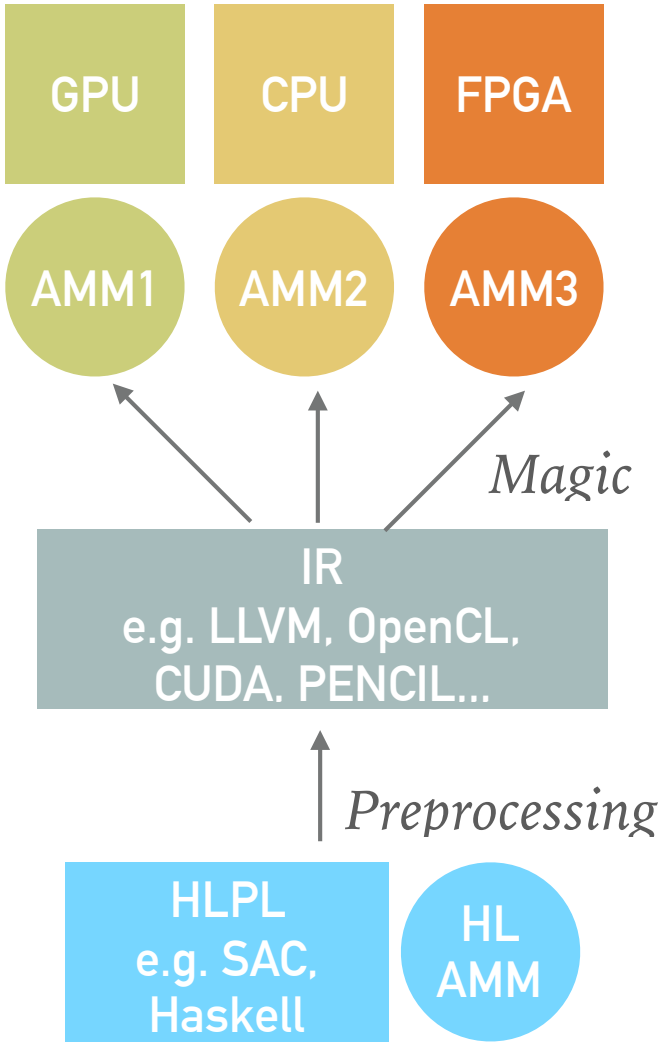


What we actually have:



WISHFUL THINKING VS. REALITY

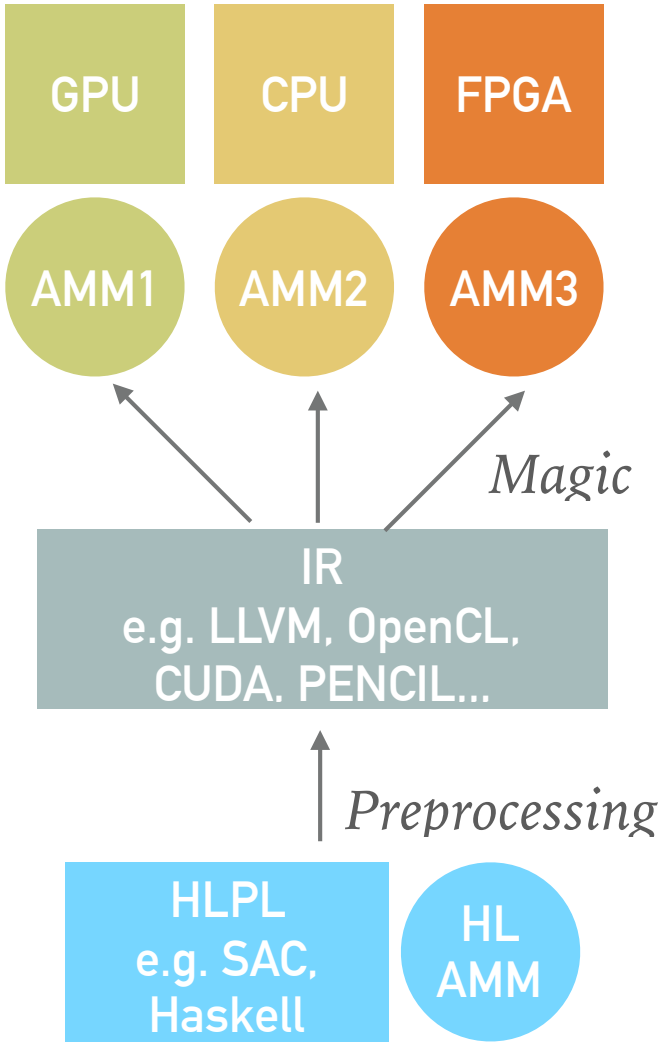
What we think we have:



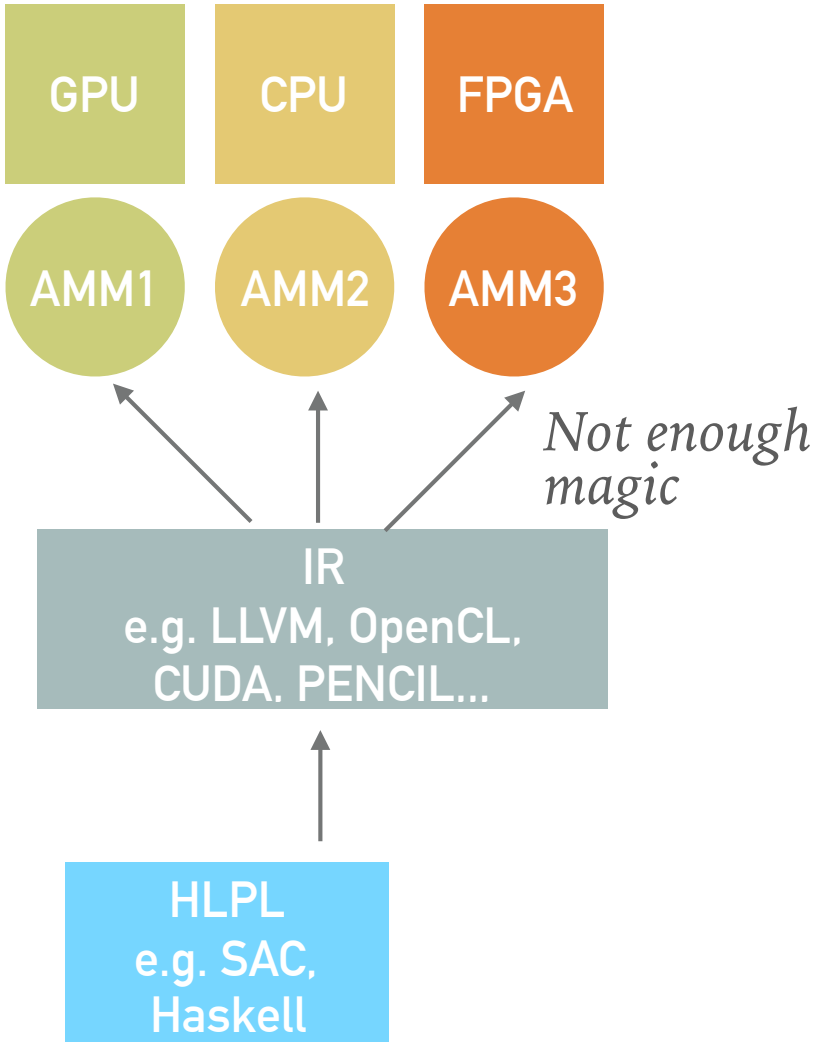
What we actually have:

WISHFUL THINKING VS. REALITY

What we think we have:

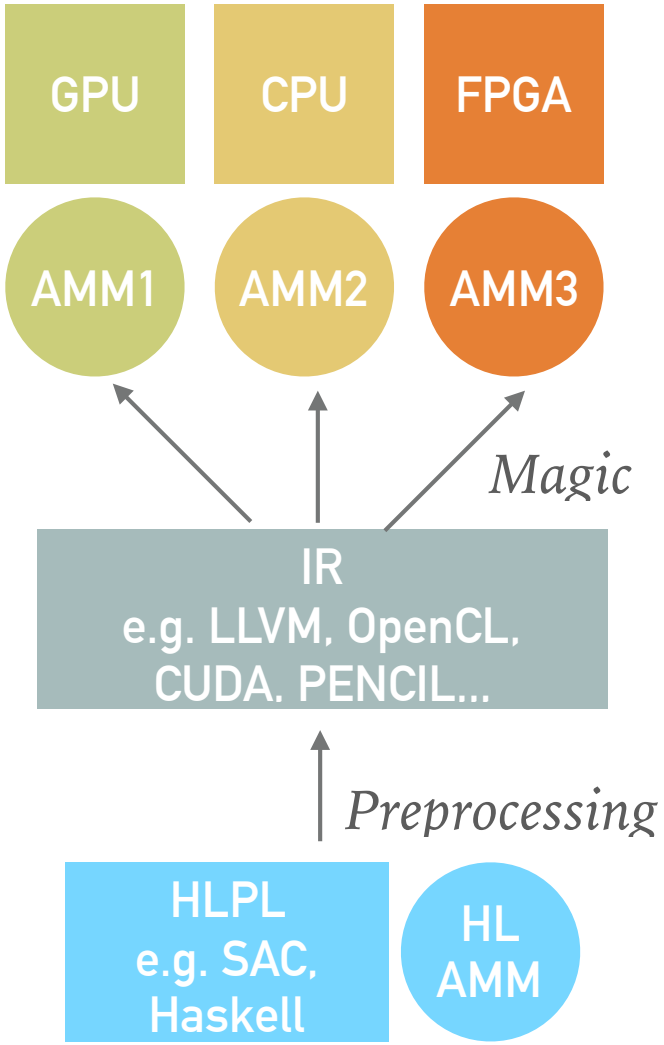


What we actually have:

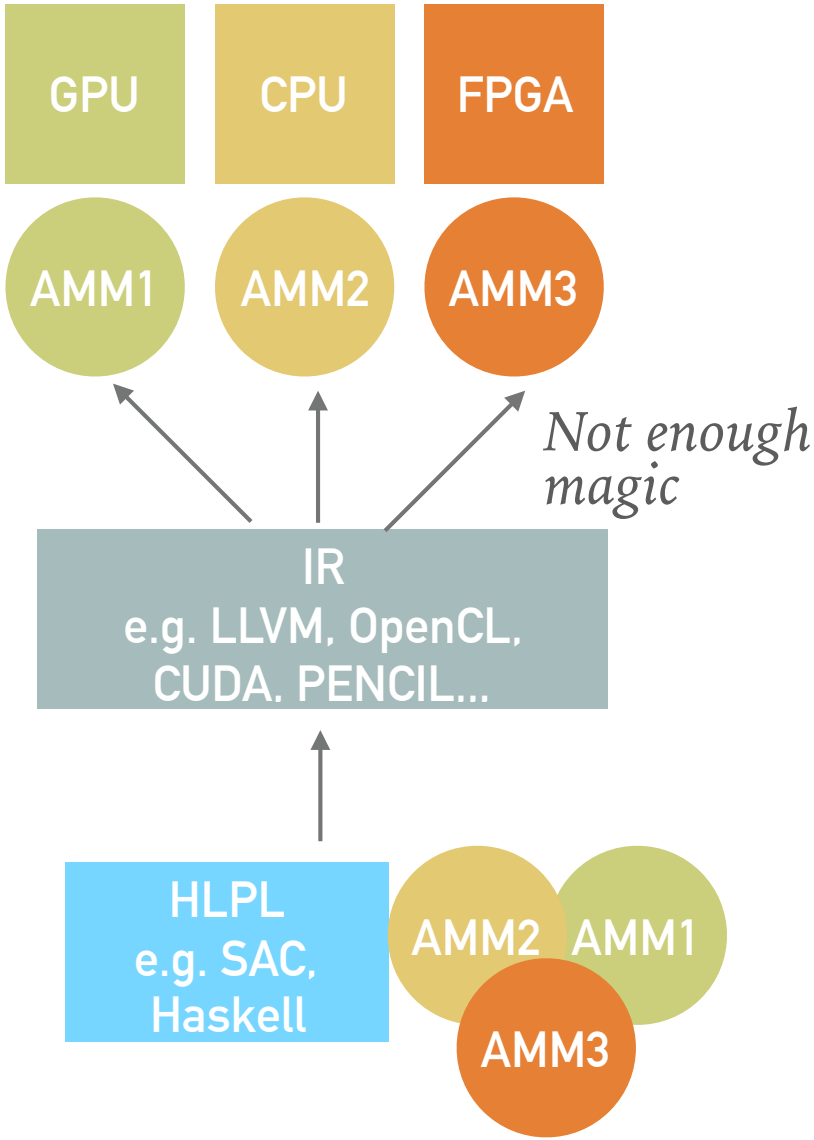


WISHFUL THINKING VS. REALITY

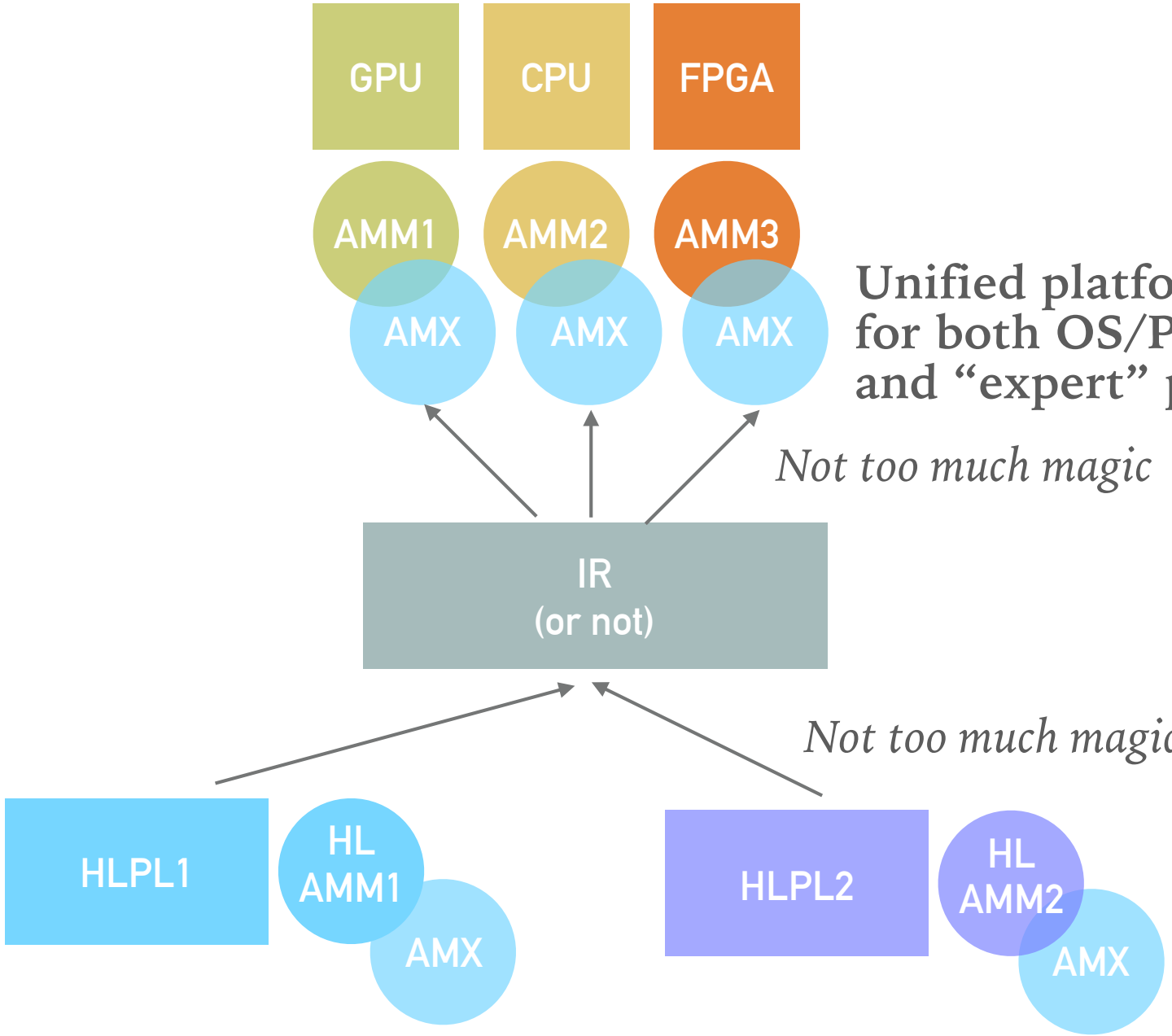
What we think we have:



What we actually have:



WHERE WE COULD GO



Unified platform AMM
for both OS/PL designers
and "expert" programmers

Not too much magic

Not too much magic

EVOLVING ABSTRACT MACHINE MODELS

- Previously:
  - **Co-evolution of AMM and architecture design independent from application code**
  - OS/PL implementers “subject” to platform AMM
  - Platform AMM seeps through app. design and crystallizes there, makes translations of code to another AMM difficult
  - Back-compatibility with previous platform AMMs paramount

EVOLVING ABSTRACT MACHINE MODELS

- Now:
  - **Diversity of HLPLs each with own operational semantics and AMM**
  - Platform AMM primary audience is OS/PL implementer, not app. dev.
  - app. dev only cares about platform AMM when overriding OS/PL intelligence
  - **Hardware vendors listen to OS/PL demands about platform AMM in new architectures (cf. GPGPUs)**
- Suggested methodology:
  Evolve arch. and AMM using HLPL design as guide!

PROGRAMMING MODELS FOR 2010–2030

PROGRAMMING MODELS FOR 2010–2030

- **Surprise observation:**
  beginner programmers are comfortable coding with ***process networks*** *(PNs) and conceptualizing / explaining data flows*
  - Suggests streaming calculus may be viable alternative to term rewriting / lambda calculus for base semantics
  - As long as PL provides powerful PN assembly combinators: people hate wiring manually

PROGRAMMING MODELS FOR 2010–2030

- **Surprise observation:**
  beginner programmers are comfortable coding with ***process networks*** *(PNs) and conceptualizing / explaining data flows*
  - Suggests streaming calculus may be viable alternative to term rewriting / lambda calculus for base semantics
  - As long as PL provides powerful PN assembly combinators: people hate wiring manually
- **Perhaps we could start with PN-based AMMs and build the rest upon this?**

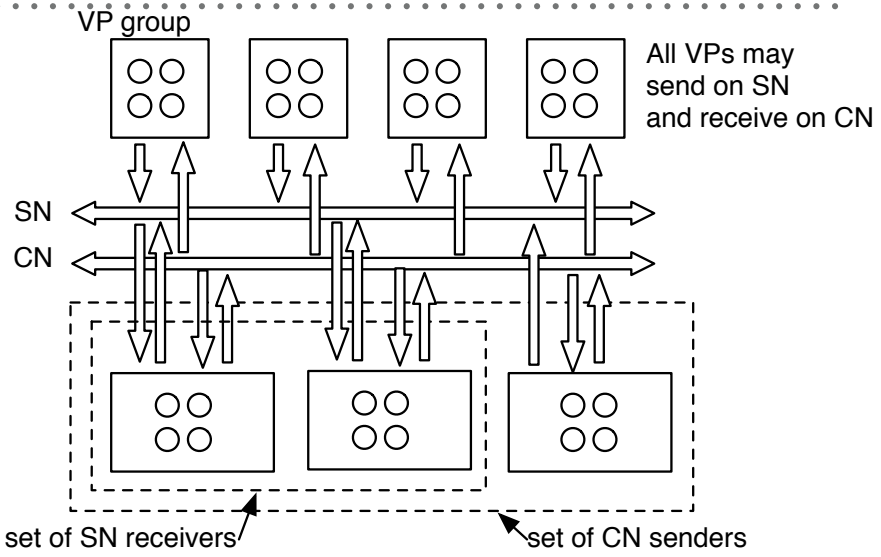
ABSTRACT MACHINE MODELS FOR PNS

- **Example: what if we build OS/PLs upon PN semantics?**
  - OK, then hardware must be good at supporting PNs
    - Surprise! already the case since ~1975 in-networks thanks to clusters and ~2005 on-chip thanks to NoCs
  - OK, then AMMs must be good at predicting behavior
    - Hmmm, VN/PRAM and DF just won't cut it. :(
    - **We need new AMMs that model network topology, interconnect protocols and asymmetry in hw platforms (now working on this with AM^3)**
  - NB: Traditional distributed computing models insufficient: new need to separate cores from memory from accelerators

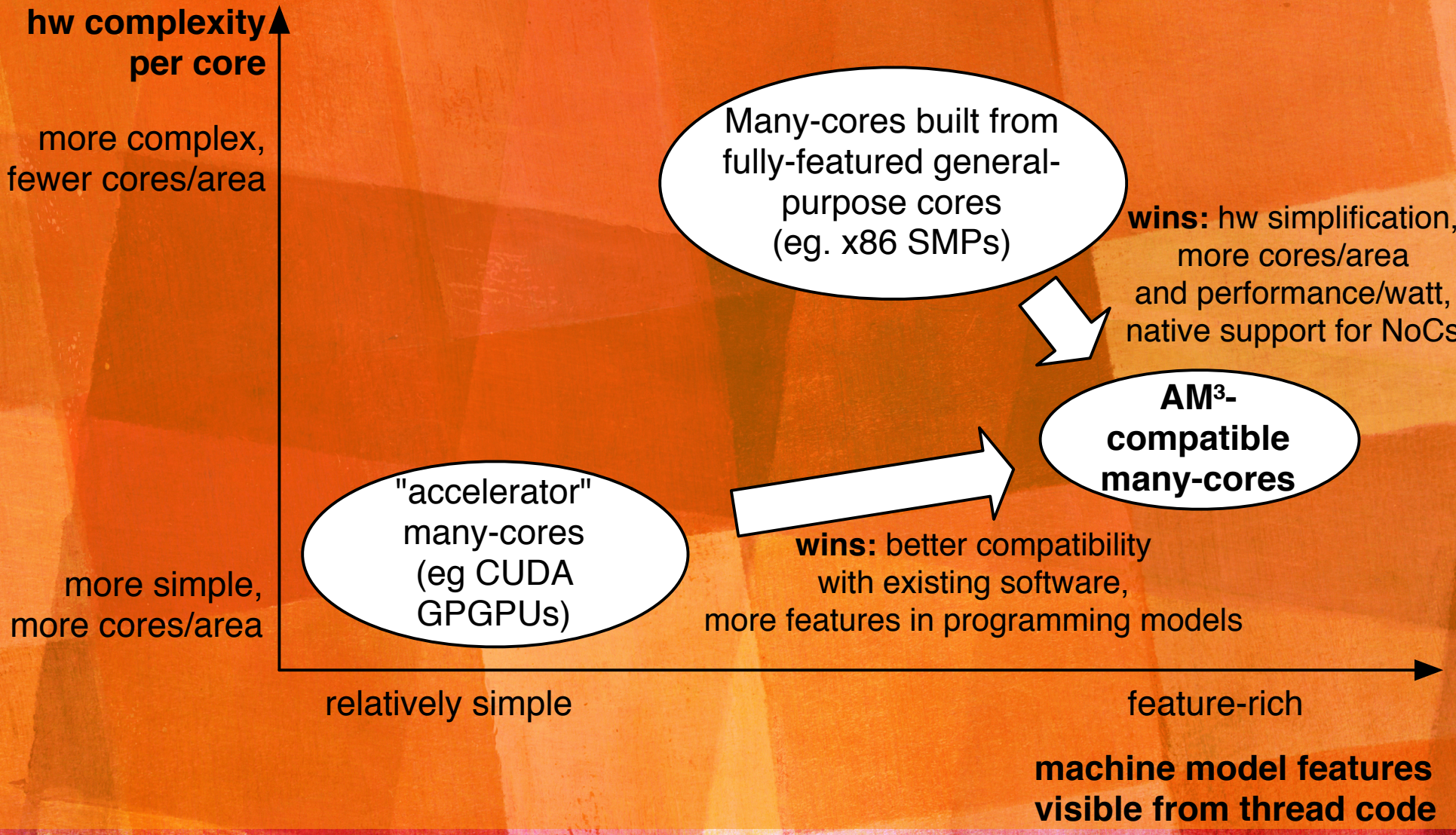
AM^3 – ABSTRACT MACHINE MODEL FOR MODERN (OR MANY-CORE) MACHINES

- **Work in progress!**

  see DOI 10.1109/TPDS.2015.2492542



- No preemption, no syscalls, no privilege separation in ISA, shared memory optional
  Network provides isolation / interfaces
- **Uniform access interface to heterogeneous compute elements**
  - Compute element = “virtual processor group”, can be heterogeneous
  - Can be CUDA warp, functional unit, hardware thread group, iRAM
- **Substitutes VN/P-RAM to implement OS or PL/RTS, sufficient to run Unix :)**


hw complexity per core
more complex, fewer cores/area
more simple, more cores/area
Many-cores built from fully-featured general-purpose cores (eg. x86 SMPs)
wins: hw simplification, more cores/area and performance/watt, native support for NoCs
AM3-compatible many-cores
"accelerator" many-cores (eg CUDA GPGPUs)
wins: better compatibility with existing software, more features in programming models
relatively simple
feature-rich
machine model features visible from thread code

OTHER AMM EVOLUTION DIRECTIONS

- **Surprise observation***: often enough, programmers do know how to optimize better than the compiler/RTS
  - PL must provide syntax/semantics with opt-in overrides
  - **HW + AMMs must provide abstractions for reasoning about (and enforcing) functional equivalence of operational mechanisms**
  - (* not only an observation, actually a theorem since 2001)
- **Empirical result 2000-2015**: people like run-time tuning!
  - **Still incomplete support in HW + AMM: how to instrument an application systematically, cross-PL and cross-platform for comprehensive run-time tuning?**

PROPOSAL: THE "BASIC" LANGUAGE OF FUTURE COMPUTERS…

1. **has a simple, easy to understand AMM that makes useful predictions on heterogeneous platforms**
2. provides main syntax to declare intent and annotations/typing to supply opt-in elidable override mechanisms
3. has processes and dataflow links as main abstract building blocks expressed using combinators, not functions, data structures, processor logic or memory contents
4. provides run-time tuning knobs separate from alg. design to:
   - ➤ trade space vs. time costs
   - ➤ trade latency vs. throughput
   - ➤ trade performance vs. accuracy/reliability/determinism/accountability
   - ➤ trade data sharing/performance vs. isolation/privacy/segregation

THANK YOU!

Questions?