An architecture and protocol for the management of **concurrency on DSoCs**

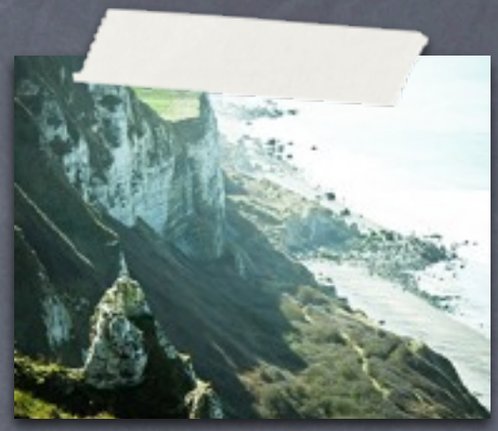
Overview - Summer 2011

Agenda

- SVP & Apple-CORE
- Realizations & evolution
- Key concepts (selected)
- Current "hot" topics
- Research directions

SVP & Apple-CORE

Origins



- 1996-present: connect many D-RISC cores together = "Microgrid"
- NWO Microgrids (2005-present): Architecture concepts, μTC and simulations
- EU ÆTHER (2007-2009): Reconfigurability and resource abstractions

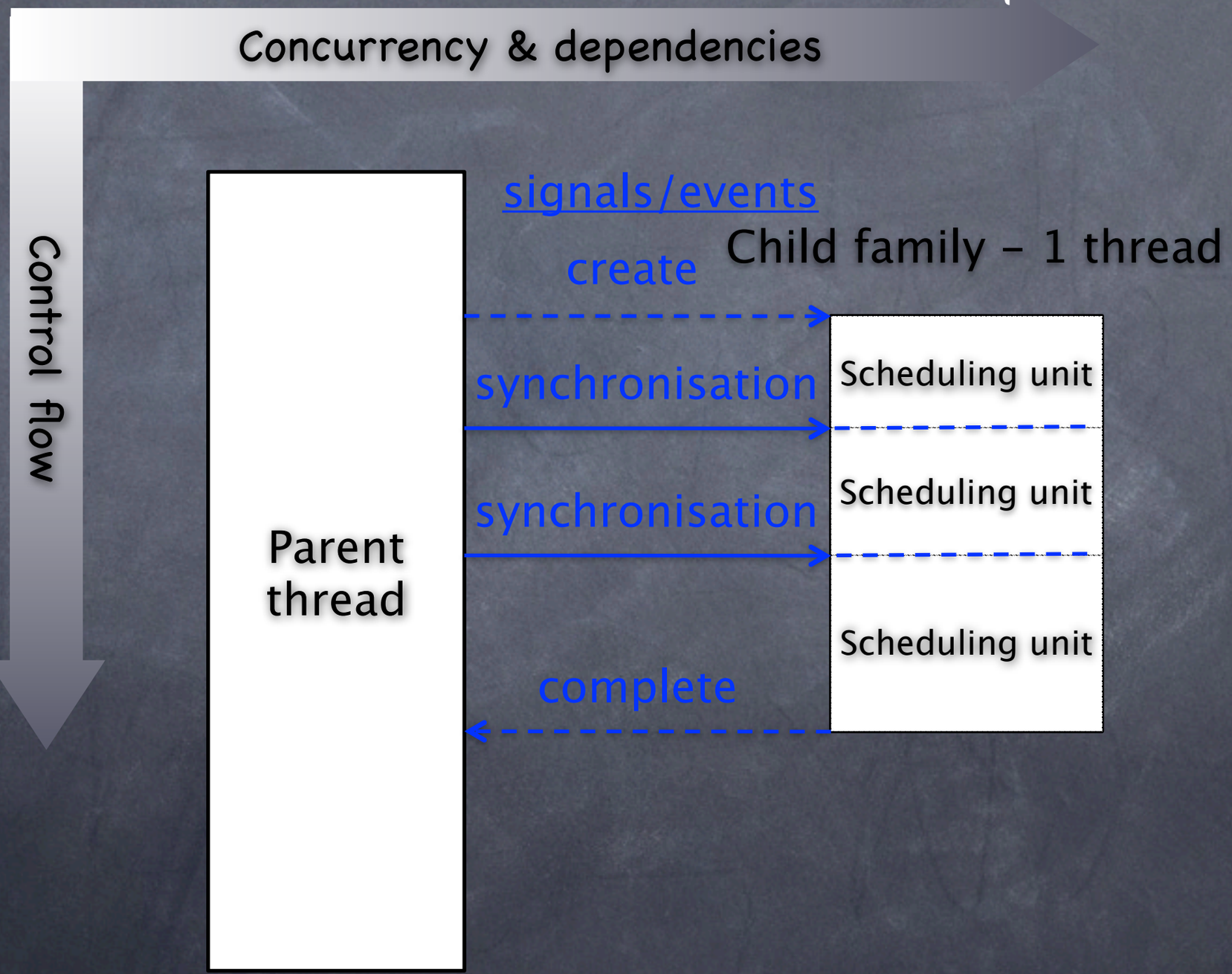
apple core

ADVANCE StatArch

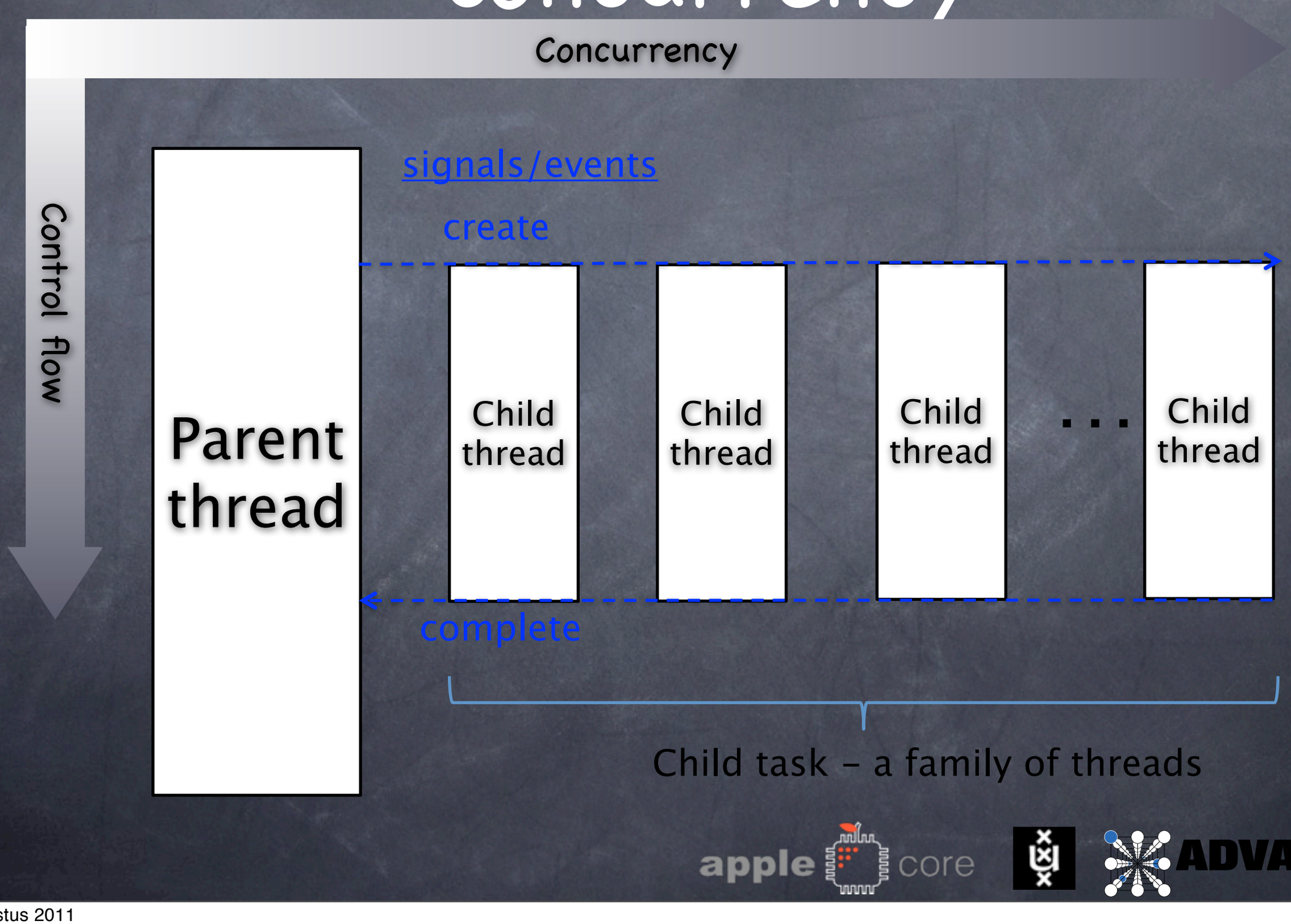
The original SVP in a nutshell

- Unit of concurrency: families of threads
- The cluster is the processor: entire families created at once over multiple cores
- Both internal and external parallelism available in hw (ILP, SMT and multi-core parallelism)
- The "create" primitive can be used in programs to define families, "sync" to bulk synchronize
- Split-phase asynchrony assumed to be the norm for all operations
- Use requires new programming model (new language, new compilation methods)

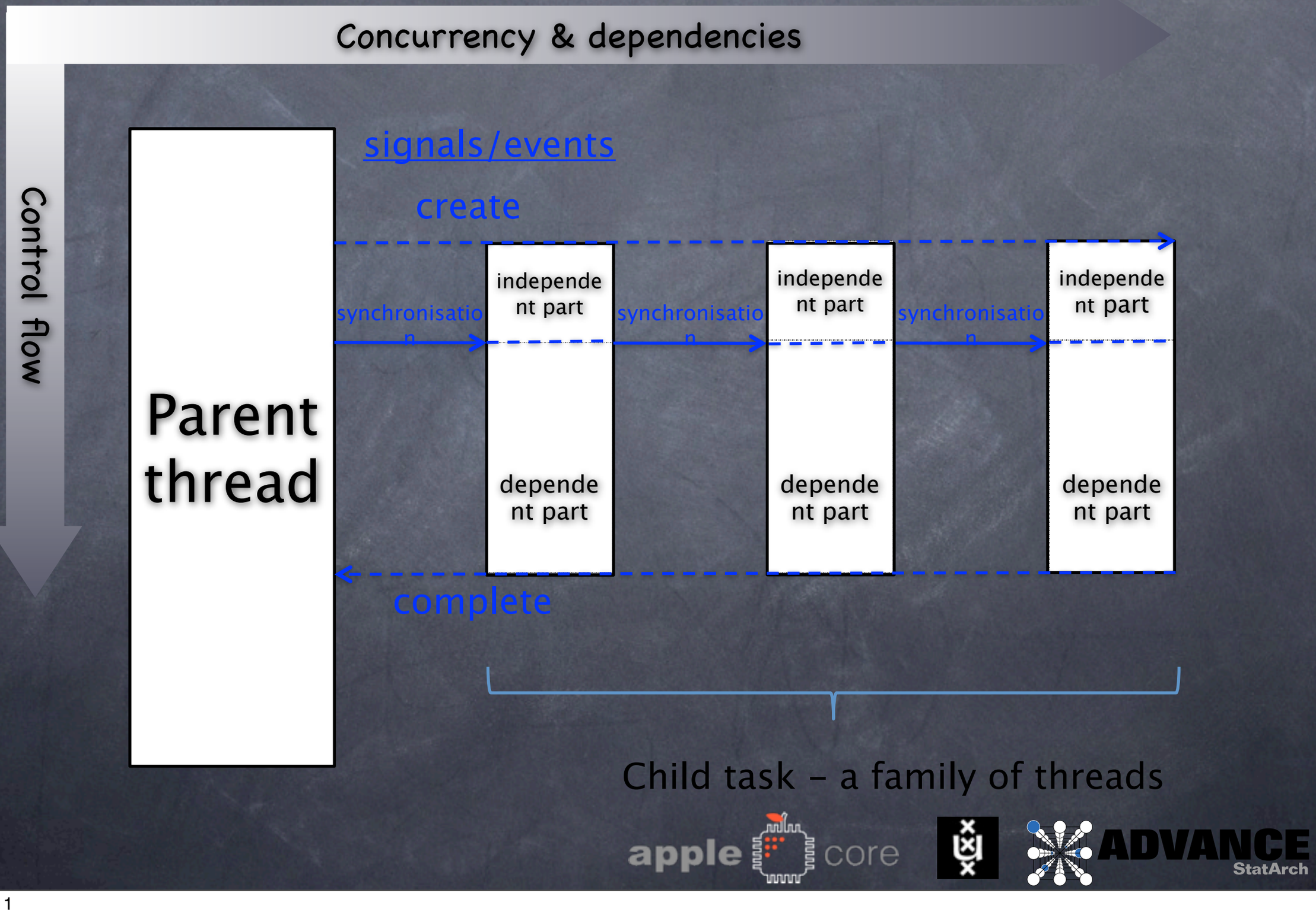
SVP functional concurrency



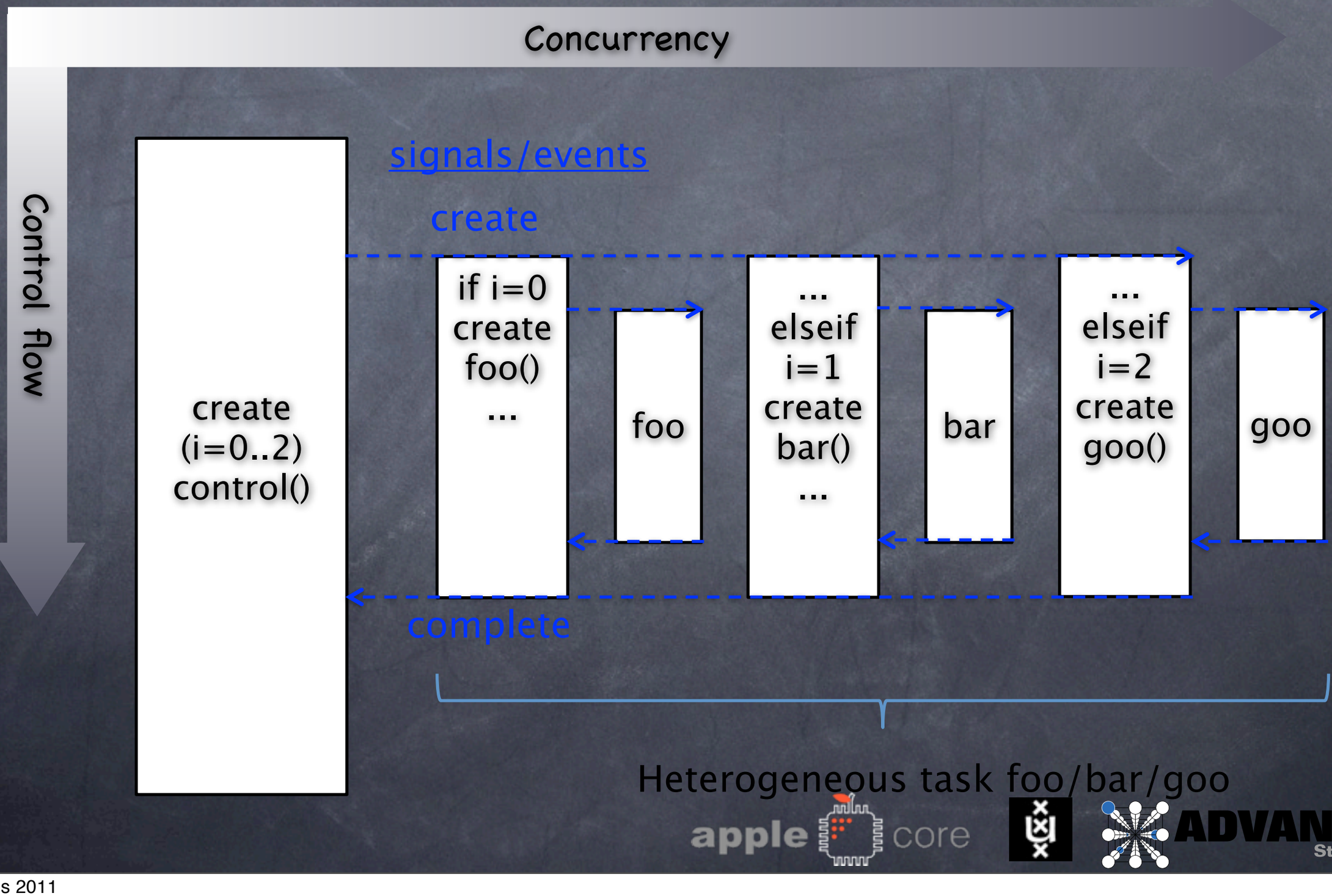
SVP replicated concurrency



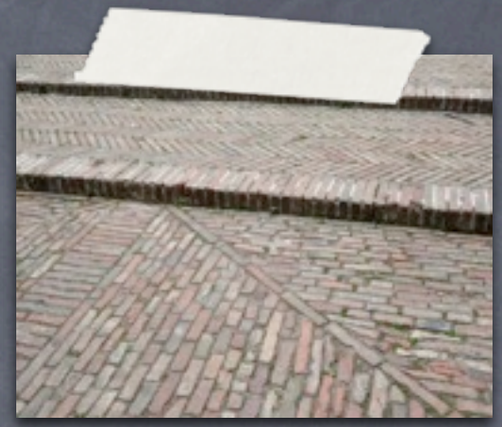
SVP Pipelined concurrency



Parent-child relationship in heterogeneous tasks



Moving up



- EU Apple-CORE (2008-2011):
  - "Can the Microgrid be programmed efficiently?"; sets μTC as an "intermediate language"
  - Goal: automatically parallelize existing code (C, SAC) while translating to μTC
  - Goal: Show applicability to real-world applications

apple core

ADVANCE StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

apple core

ADVANCE
StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Many cores

apple core

ADVANCE
StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Many cores

apple core

ADVANCE
StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Many cores

Apple-CORE strategy

Compute kernels
(performance languages)

Compute kernels
(retargetable concurrent SVP IR)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Many cores

apple core

ADVANCE
StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Compute kernels
(retargetable concurrent SVP IR)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Many cores

apple core

ADVANCE
StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Compute kernels
(retargetable concurrent SVP IR)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Many cores

apple core

ADVANCE
StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

parallelizing
compilers

Compute kernels
(retargetable concurrent SVP IR)

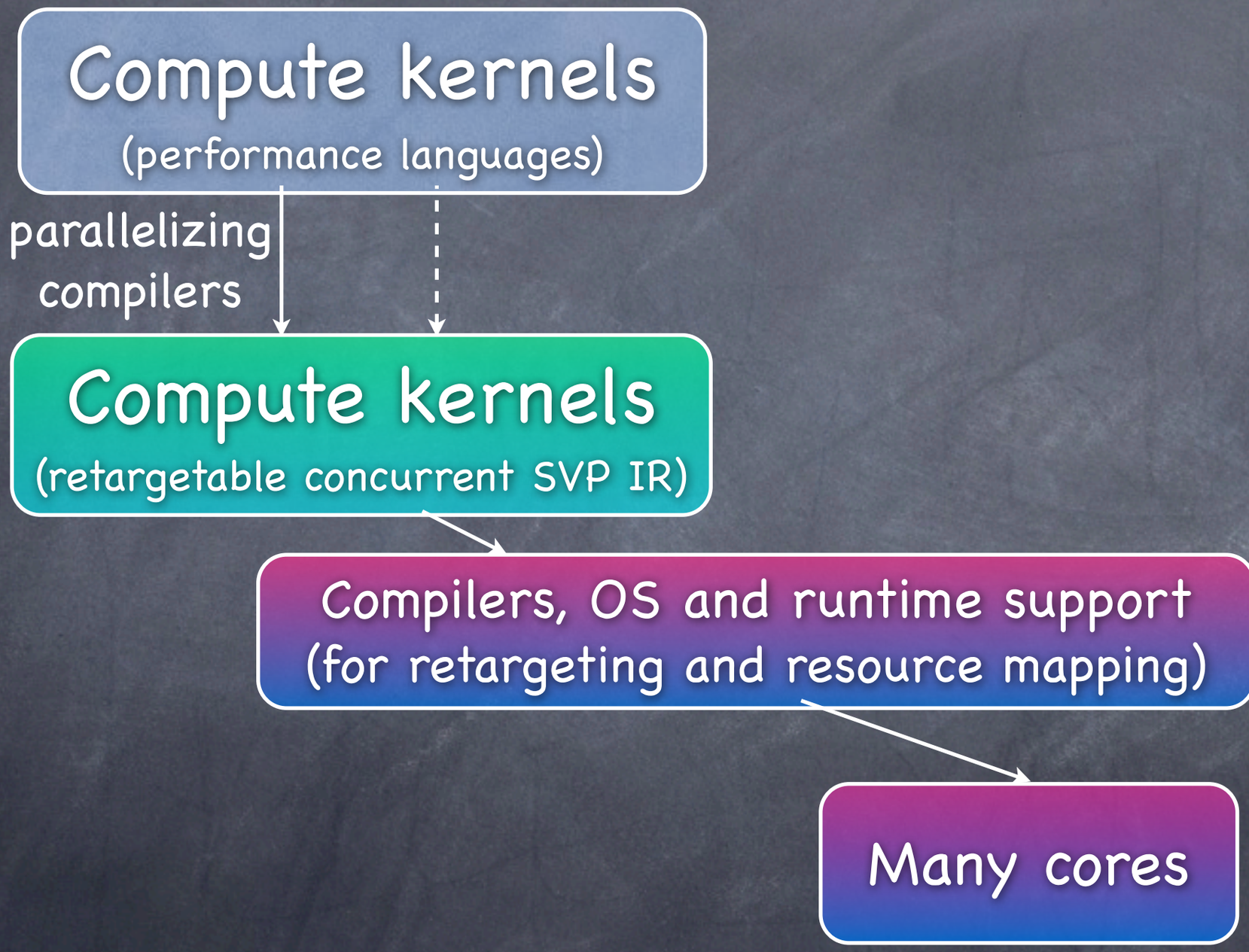
Compilers, OS and runtime support
(for retargeting and resource mapping)

Many cores

apple core

ADVANCE
StatArch

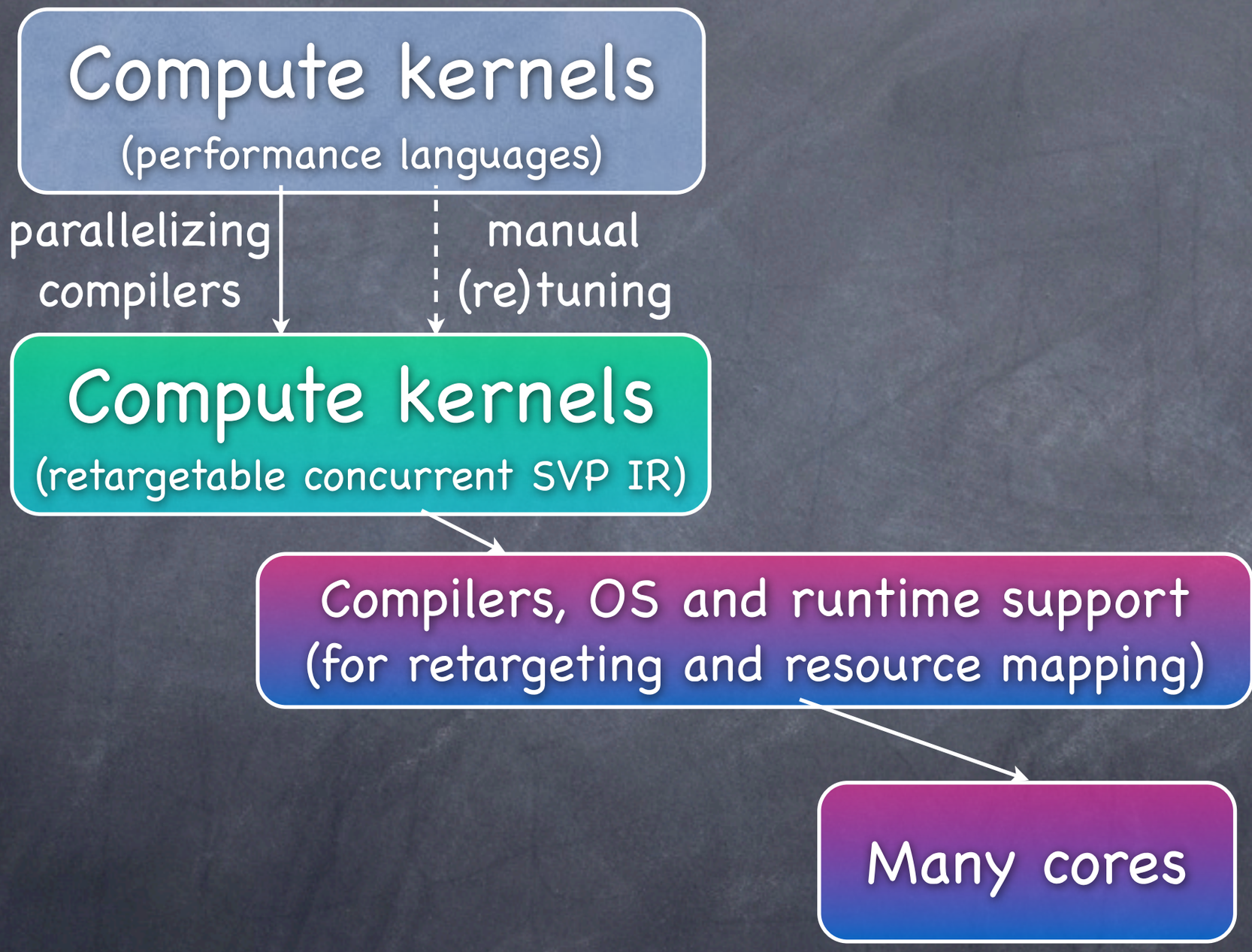
Apple-CORE strategy



apple core

ADVANCE
StatArch

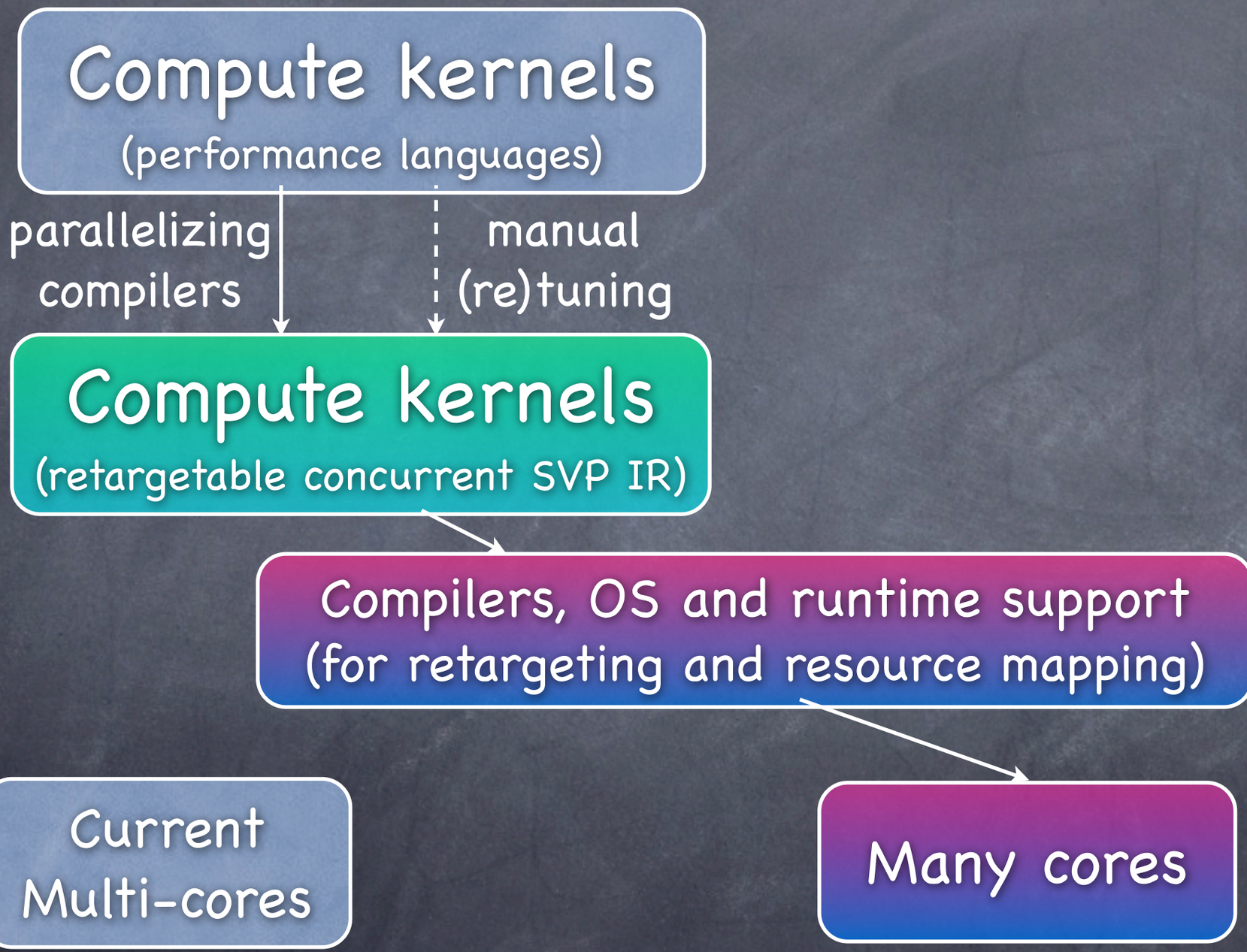
Apple-CORE strategy



apple core

ADVANCE
StatArch

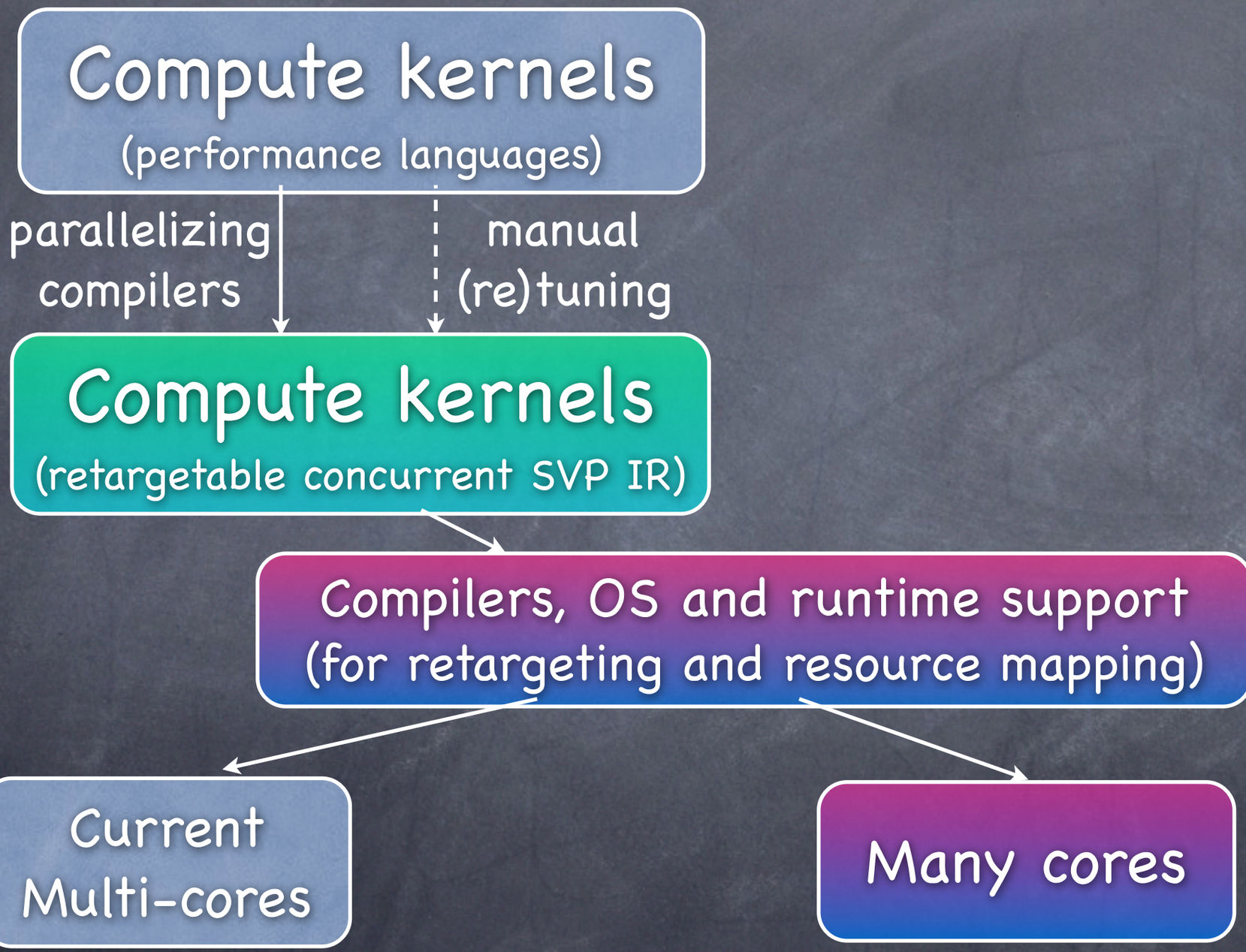
Apple-CORE strategy



apple core

ADVANCE StatArch

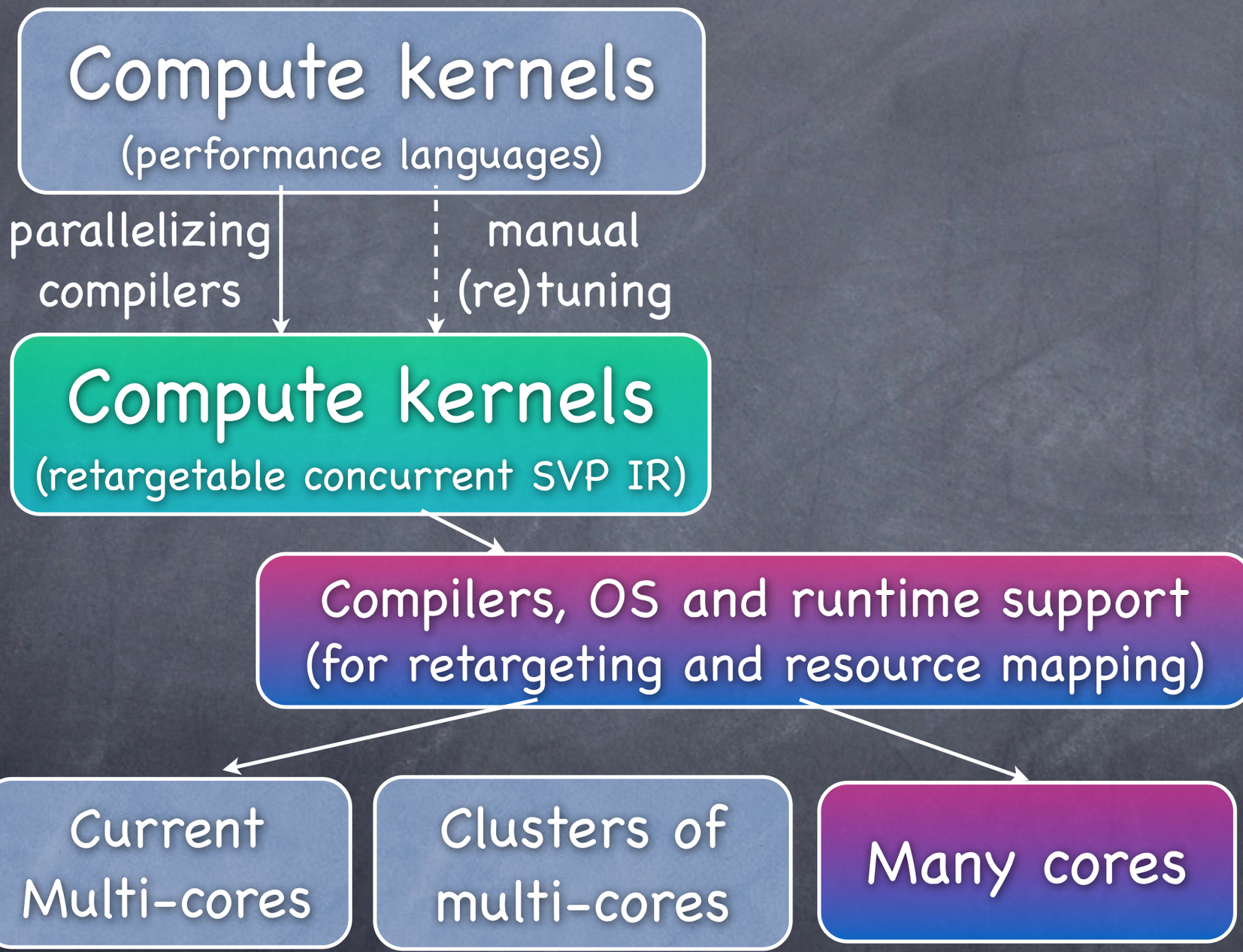
Apple-CORE strategy



apple core

ADVANCE StatArch

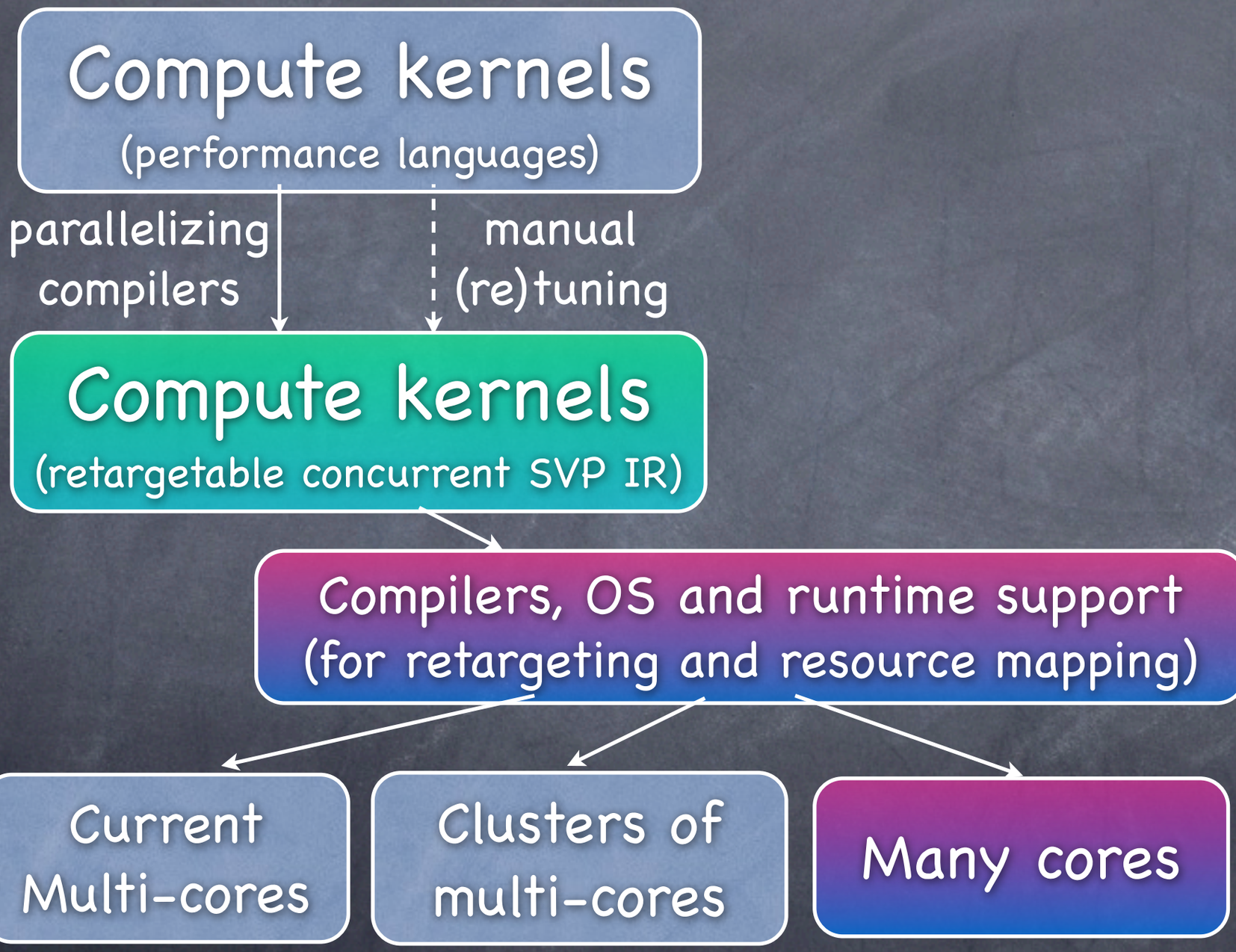
Apple-CORE strategy



apple core

ADVANCE StatArch

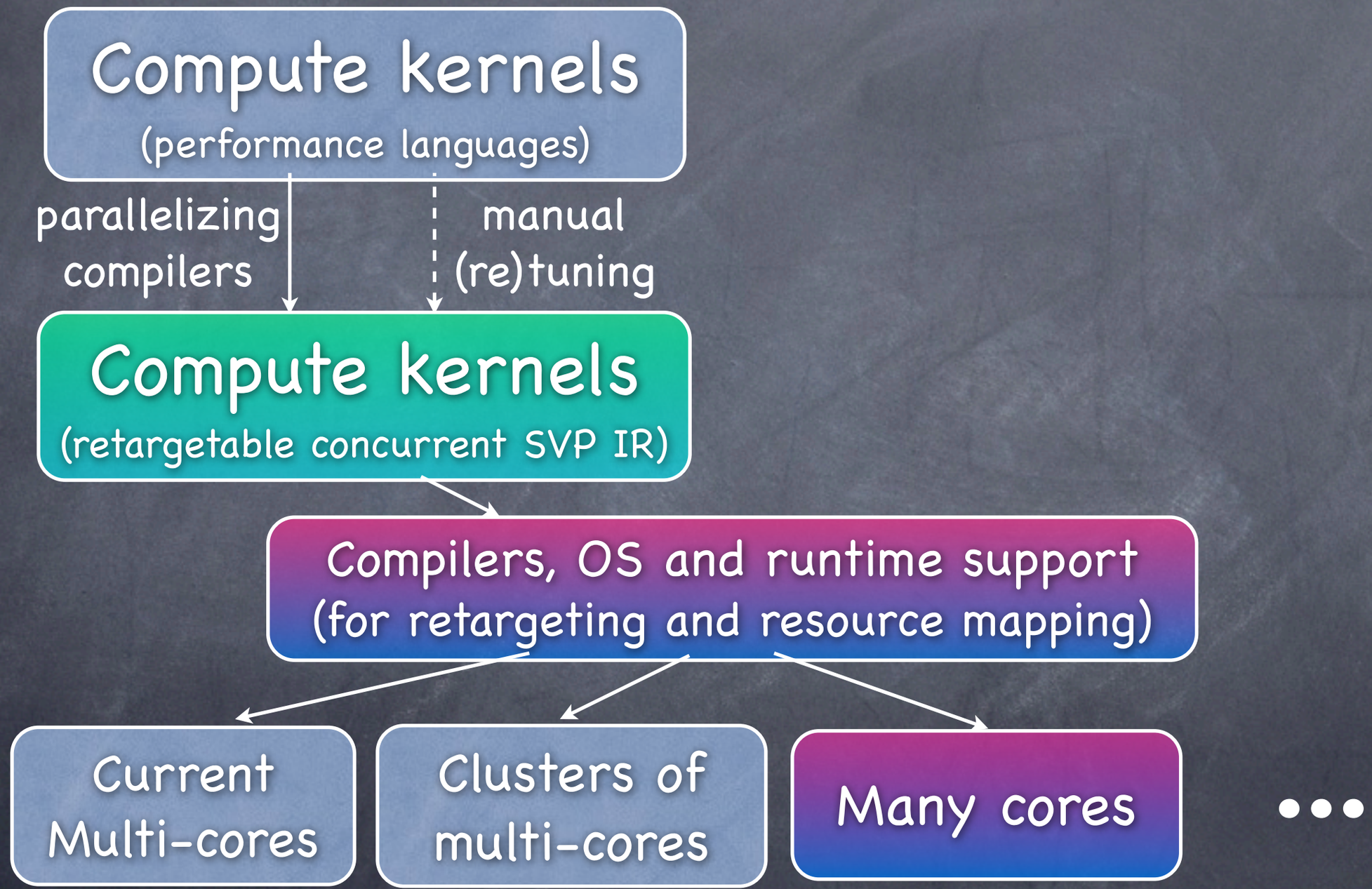
Apple-CORE strategy



apple core

ADVANCE StatArch

Apple-CORE strategy



apple core

ADVANCE StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Program assemblies
(composition languages)

parallelizing compilers

manual (re)tuning

Compute kernels
(retargetable concurrent SVP IR)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Current Multi-cores

Clusters of multi-cores

Many cores

•••

apple core

ADVANCE StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Program assemblies
(composition languages)

parallelizing compilers

manual (re)tuning

Compute kernels
(retargetable concurrent SVP IR)

Program assemblies
(SVP-aware composition languages)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Current Multi-cores

Clusters of multi-cores

Many cores

•••

apple core

ADVANCE StatArch

Apple-CORE strategy

Compute kernels
(performance languages)

Program assemblies
(composition languages)

parallelizing compilers

manual (re)tuning

Compute kernels
(retargetable concurrent SVP IR)

Program assemblies
(SVP-aware composition languages)

Compilers, OS and runtime support
(for retargeting and resource mapping)

Current Multi-cores

Clusters of multi-cores

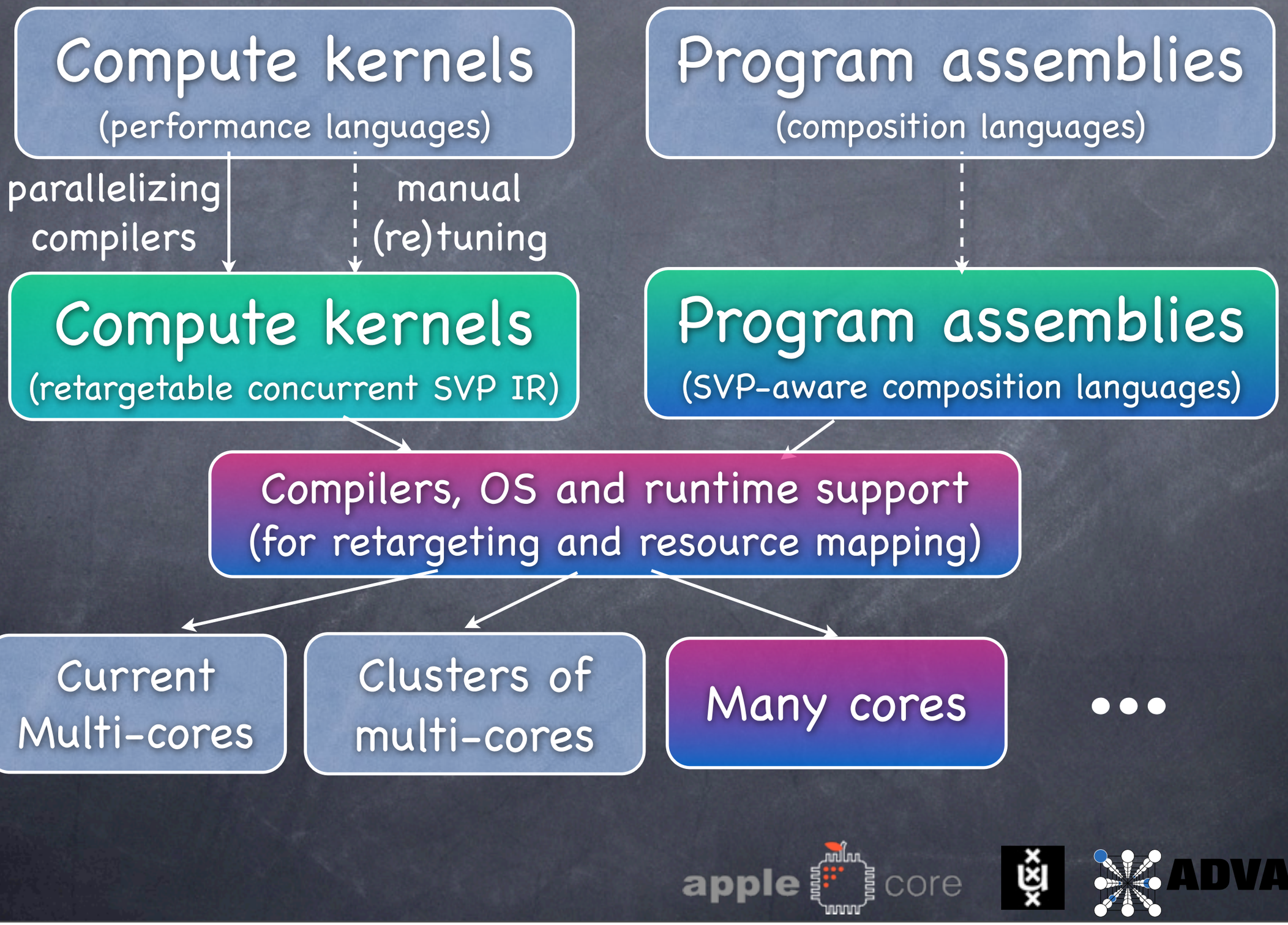
Many cores

•••

apple core

ADVANCE StatArch

Apple-CORE strategy



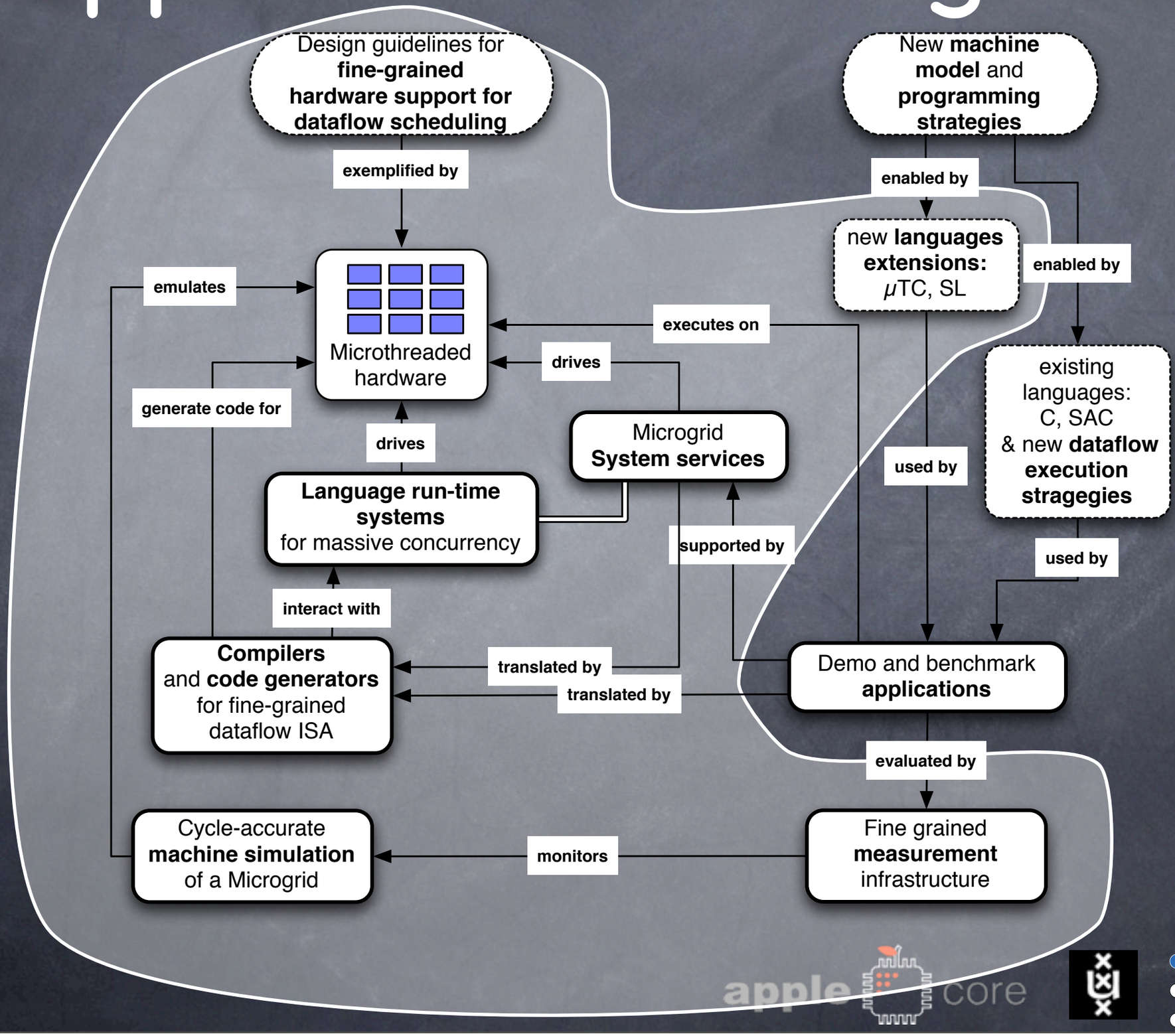
Duality in SVP (original view)

- "SVP is dual, a combination of a programming model (provides concepts and contracts) and an architecture model (provides abstract mechanisms)"
- The Microgrid and μTC-ptl (higher-level simulation) are implementations of SVP

Moving up (cont.)

- Cleaning up (2008-2009): Programming and architecture models stabilized, described and specified separately
- Beefing up (2009-2010):
  - Compiler toolchains actually useable
  - OS+compiler support for resource deadlocks, mutual exclusion, arbitrary inter-thread communication

Apple-CORE foreground



Apple-CORE results

- Concurrency within cores provide good latency tolerance - assuming enough threads

- Distribution of work across cores is technically easy and can be achieved at extremely low latencies (few cycles)

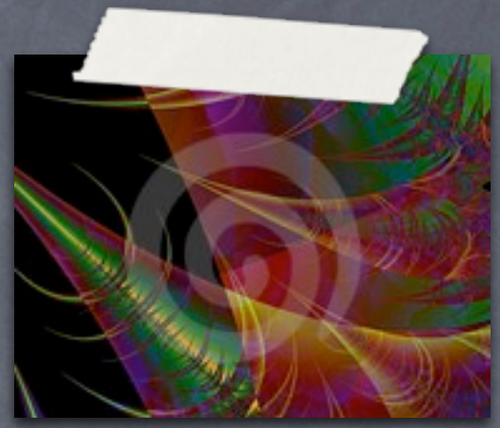
  - the difficult problem is placement decision

- We have designed a full chip architecture
  = custom in-order microthreaded RISC cores on a custom NoC with custom memory network

apple core ADVANCE StatArch

Pause and reflection

Realizations



- There is more to a **usable system** than a processor (or network thereof) and code generator
- Embedding dataflow + microthreading into a core is a disruptive architectural change

  - **too easy to under-estimate compilation issues**
  (μtc was a management failure, had to re-design mid-project)

- The interface between hardware and software is not only a matter of language, OS syscalls and standard user library

  - **too easy to under-estimate integration issues**
  (an SVP-only system prevented reuse of benchmark code)

Realization 1

"Today"

Multiple sequential applications

Extend languages

Coarse time sharing software scheduler

Shift responsibility

Single processor

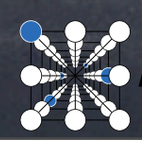
Multi-granularity concurrent code

Hardware concurrency & scheduling

Many cores

"Tomorrow"

Future architectures



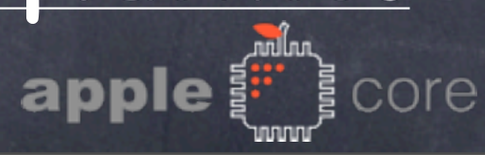
ADVANCE
StatArch

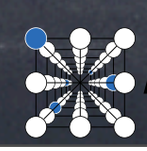
Realization 1

Today:

Multi-tasked applications	Time and space sharing schedulers	Multiple cores
Language inter-operability	Standard system services	Memory hierarchies
Portability, compatibility	Device drivers	I/O devices

Expectations **Standard practices**





ADVANCE
StatArch

Realization 2

Assembler

Insn matching +
Regalloc +
Static scheduling

Linker

Language
front-ends

Device
Drivers

Driver
interfaces

Language
libraries

Hardware
guy

Language
and library
documentation

Software
guy

Introspection
interfaces

Resource
managers

System
documentation

Machine
access
interfaces

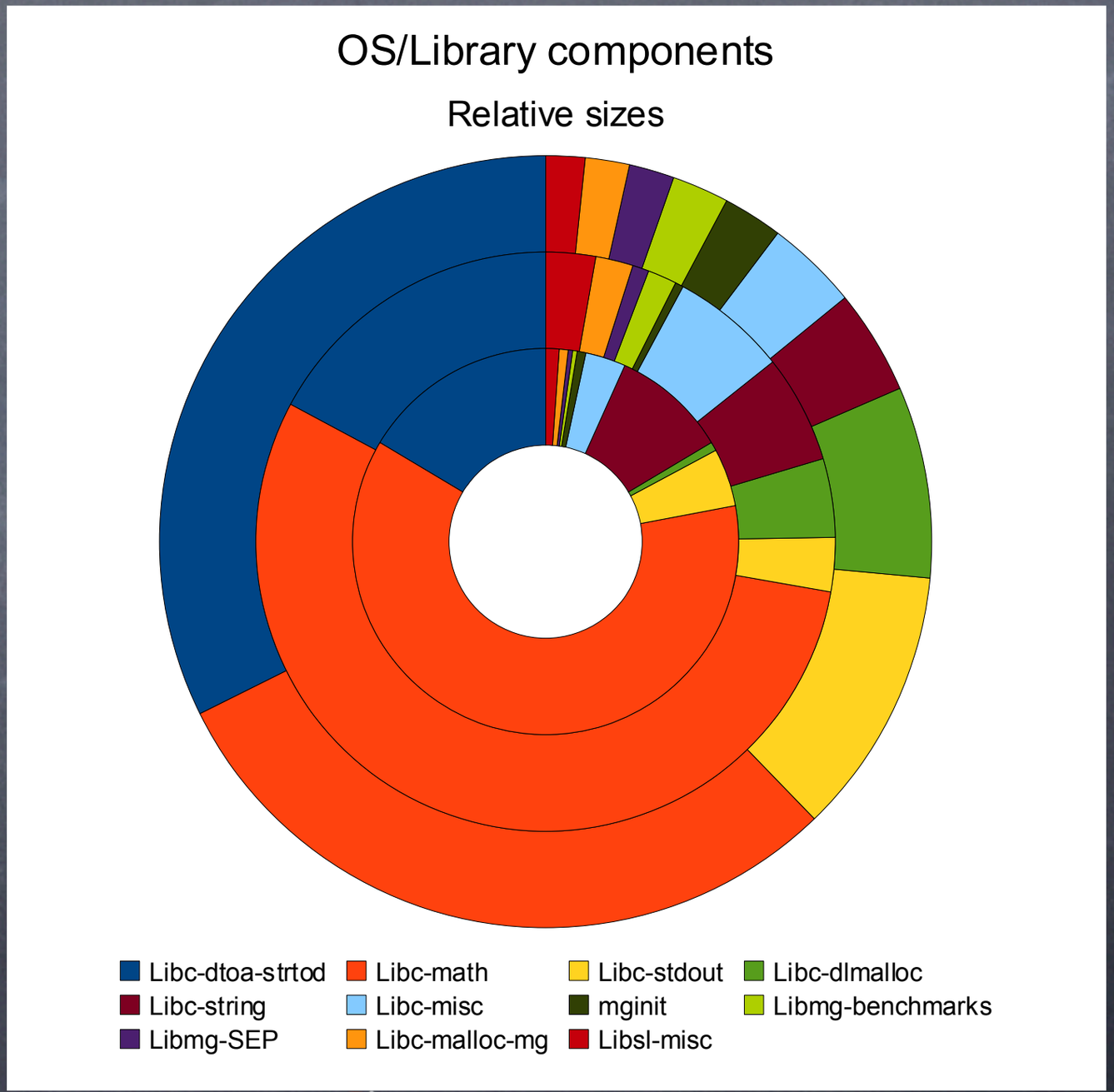
Tool chain
packaging

Debuggers

Shells and all other system and user
tools and utilities

Evidence from the A-C experience

Component	Objects	Functions	Code words
Libc-dtoa-strtod	44	74	14528
Libc-math	165	238	13424
Libc-stdout	13	13	5040
Libc-dlmalloc	2	19	3600
Libc-string	26	26	1968
Libc-misc	9	28	1728
mginit	2	2	1120
Libmg-benchmarks	1	7	1072
Libmg-SEP	1	4	848
Libc-malloc-mg	2	9	832
Libsl-misc	3	12	736



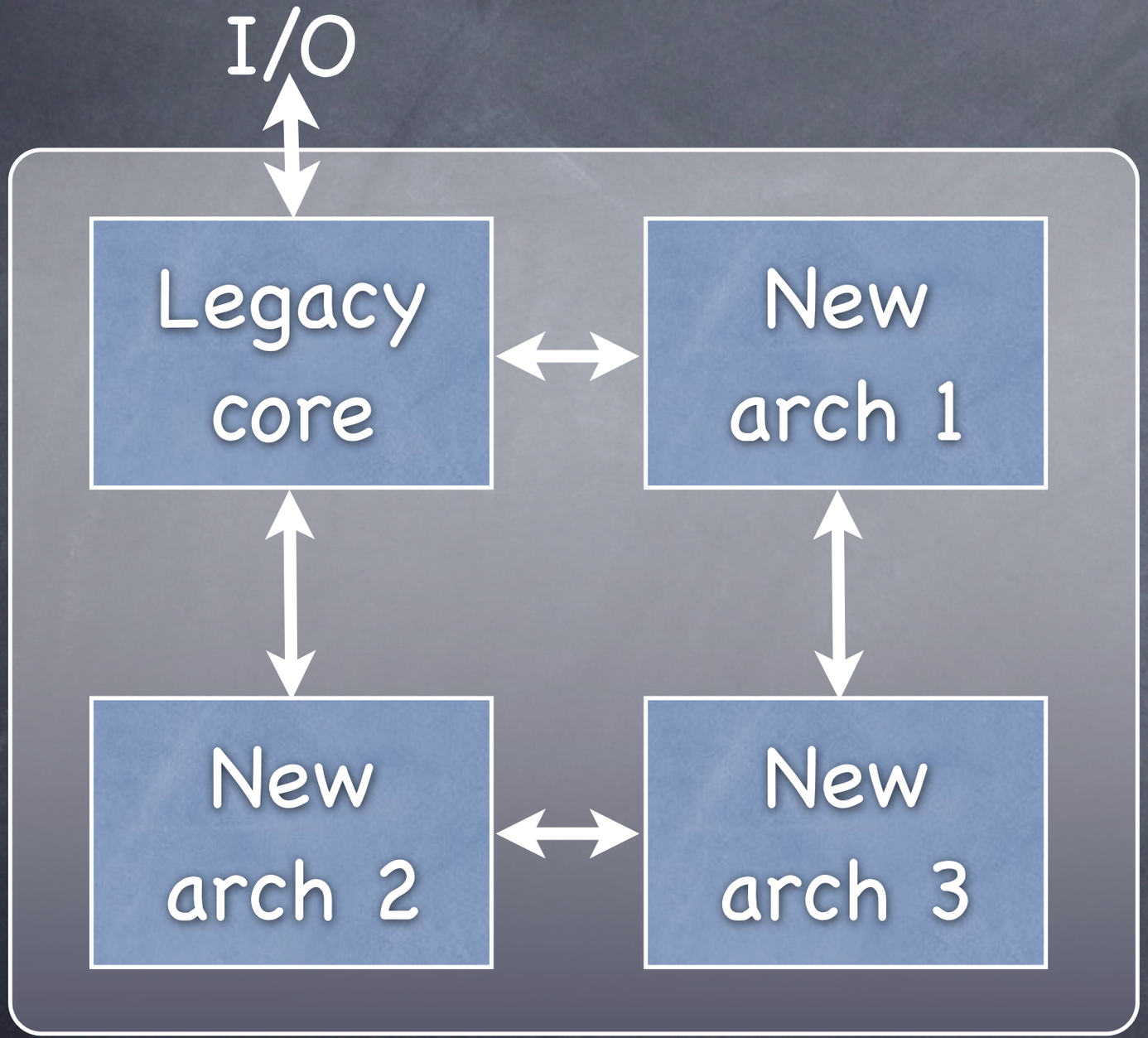
Realization 2

- **Linguae francae**
  - C/POSIX
    (not your textbook C: inline assembly, compiler extensions)
  - Java/JVM
  - CIL/CLR (.NET)
- Everything else either uses or implements these
- Pick your lingua franca = pick your community


How to pitch novel architectures?

- (Assuming the hardware is ready for use)
  - Implement a compiler for a **restricted dialect of C** with **minimal library support**
    - tractable, but expect: "meh... what can you do with it?"
    - the bane of architecture research: underestimate sw integration
  - Implement a **complete GCC target** then port an **existing operating system** with bootloader, hw drivers, file system, shell and utilities
    - untractable, unless new hw ≈ old hw (to maximize software reuse)
- What else?

Heterogeneous systems



- Run the linguae francae on an existing design
- Embed the existing design as component of a D-SoC
- Define **standard protocols** for control and communication between cores

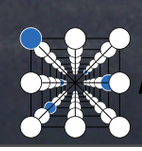
Evolution / transition

- Shift from core architecture design
  to **system design and integration**
- Innovate with **protocols and language integration**
  instead of machine primitives and programming models
  (hint: tcp/ip too heavy for low-latency on-chip interactions)
- New pitch:

  "SVP is a protocol and architecture for the management of execution units on a chip"

Proposed base semantics

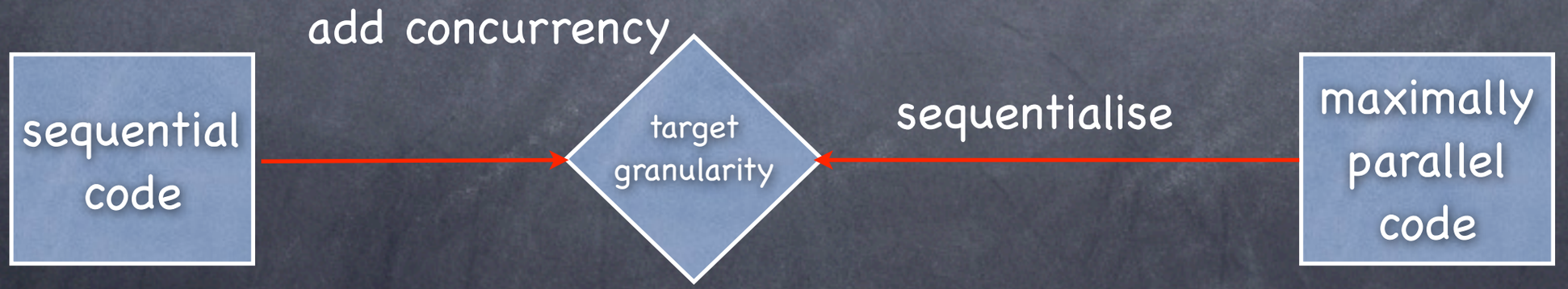
- **Dataflow synchronization cells** to support split-phase asynchrony between threads (int. concurrency) and components (ext. concurrency)
- NoC messages between and within cores with fixed semantics:
  - **allocate/release** to manage execution resources
    - **create** to trigger execution of a family
    - **sync** to bulk-synchronize a family
    - **rput/rget** to access synchronizers remotely
- Multiple ISA integration strategies: adaptation, extension, embedding
- Responsibility for consistent use pushed up to the software stack, not constrained by model any more

Key concepts (selected)

Granularity

- Code portability = distributing different computation grain sizes
- execution efficiency proportional to grain size, inversely to cost of managing concurrency
- Also have to consider the communication cost of distribution

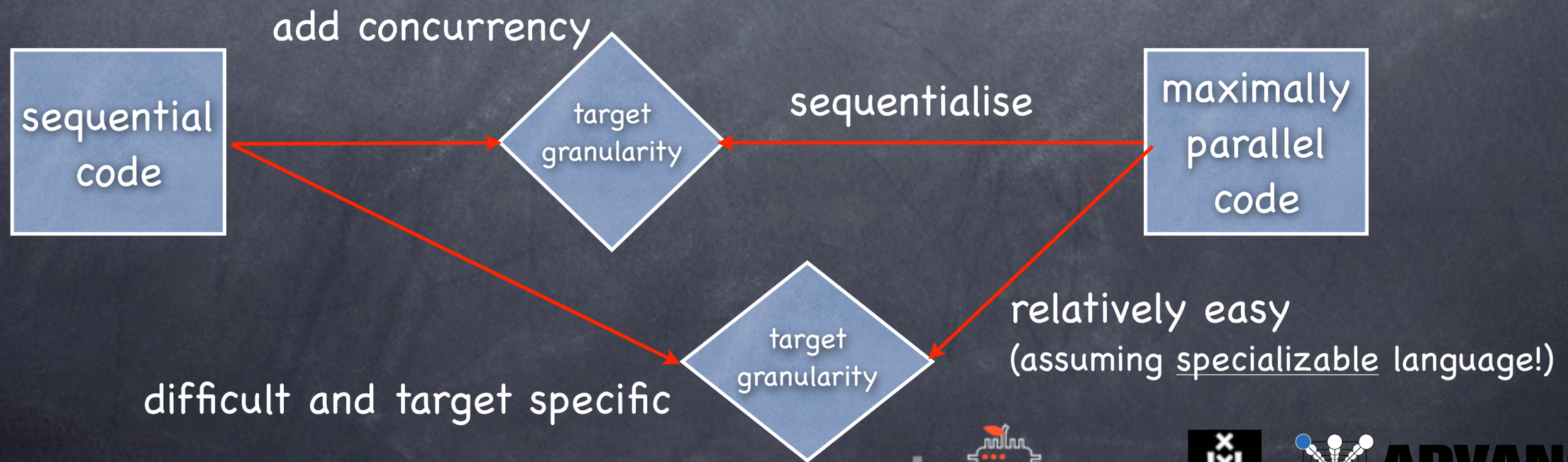


difficult and target specific

relatively easy
(assuming specializable language!)

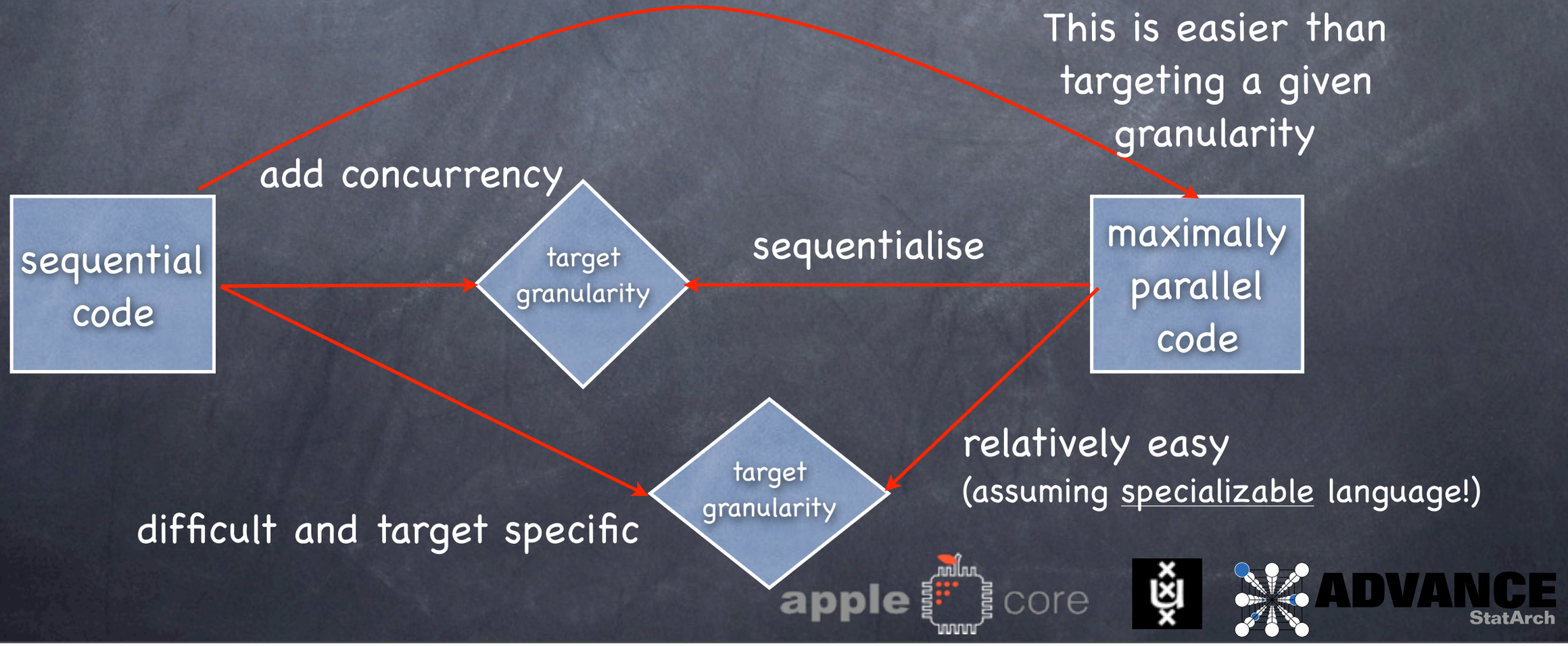
Granularity

- Code portability = distributing different computation grain sizes
- execution efficiency proportional to grain size, inversely to cost of managing concurrency
- Also have to consider the communication cost of distribution



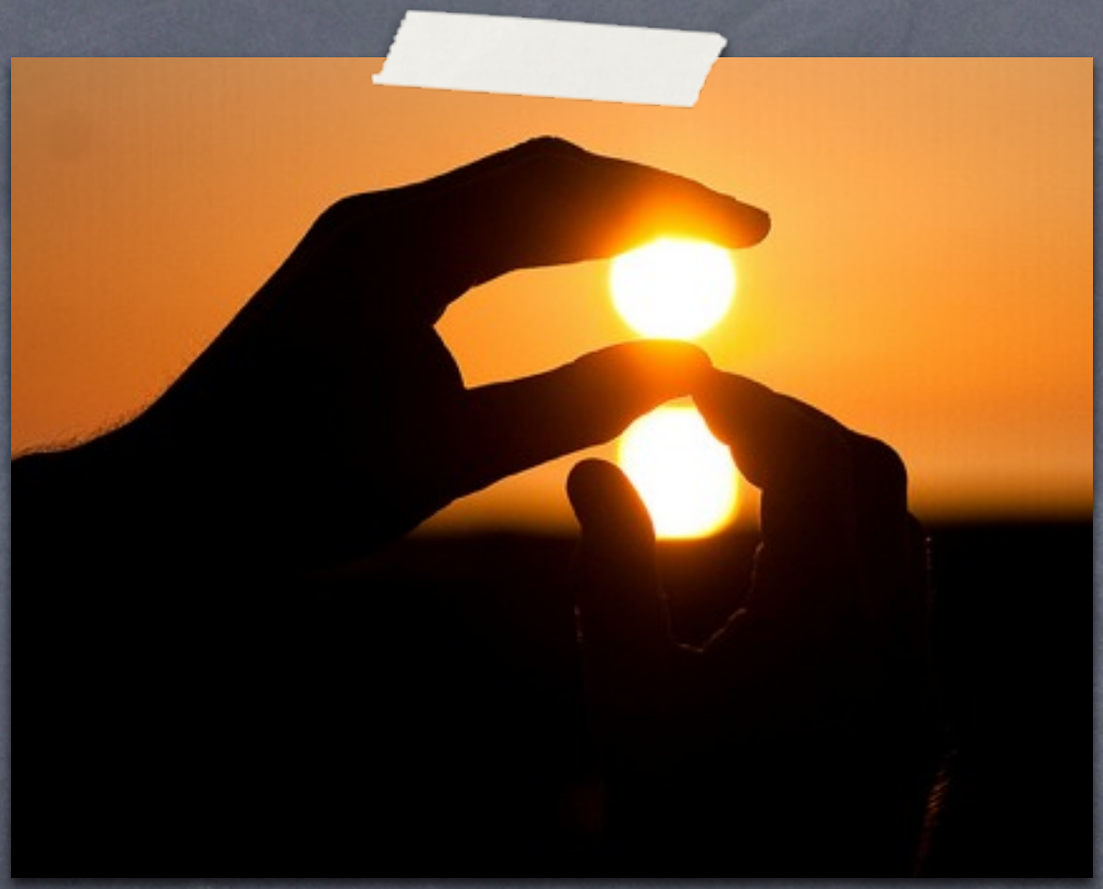
Granularity

- Code portability = distributing different computation grain sizes
- execution efficiency proportional to grain size, inversely to cost of managing concurrency
- Also have to consider the communication cost of distribution



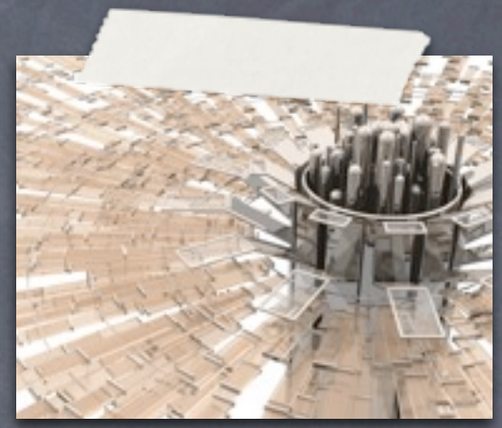
Mutual exclusion

- Mutual exclusion is required to support controlled non-determinism
  - DSoC memory is asynchronous, only weakly consistent might not be usable to implement atomics
    - the SVP way is to define a "mutex place" that serialises all requests to create tasks at that place (Dijkstra's secretaries) - same protocol
- These can be used for multiple synchronization goals, e.g. I/O, semaphores, resource allocation, etc.
- As such they are in the domain of systems programming i.e. the concurrency engineer

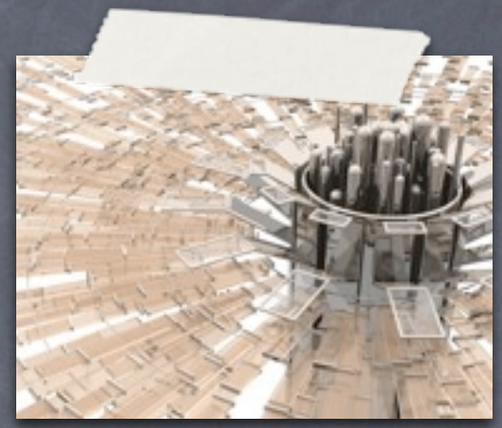
Current "hot" topics

Inter-thread communication



- Communication channels beween threads:
  - OK for register-sized channels: use i-structure hardware registers
  - Not OK for larger items: they must fit through memory ... which is asynchronous
  - Which memory model? How to use it in programs? Non-shared address spaces?

Inter-thread communication



- Some directions:
  - Programming model: make all communication visible to the compiler
  - Implementation using shared memory: synchronize memory updates via registers
  - Distributed systems: explicit messaging

Resource deadlocks

- Shortage of memory: not our problem (yet)
- Shortage of synchronizers: spread threads on multiple cores - OK
- Shortage of thread entries: create less threads per core - OK
- Shortage of family entries: delegate ... where to? or sequentialize ... how?

Resource deadlocks

- Some directions:
  - Rebalance concurrency trees early (by rewriting code for family creates)
  - Delegation (to other places on the SoC)
  - Automatic sequentialization, requires both:
    - feedback from hardware about resource availability
    - sequentializability of concurrency pattterns in languages

Research directions

Follow-ups to Apple-CORE

- How to **integrate the protocol** in existing and future core architectures / ISAs

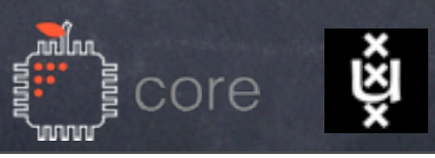
  - in a way that minimize disruption in software stacks?

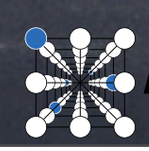
- Is there a case for **multiple ISAs** in the same chip

  - and how to manage multiple codes in the same program?

- How to **automate placement** over multiple cores

  - with a placement latency comparable to (or lower than) the operation latency?





Software engineering ADVANCE

- "What information should be available in programs, and how to represent resources, to allow efficient mapping and scheduling to computing resources?"

apple core

ADVANCE StatArch

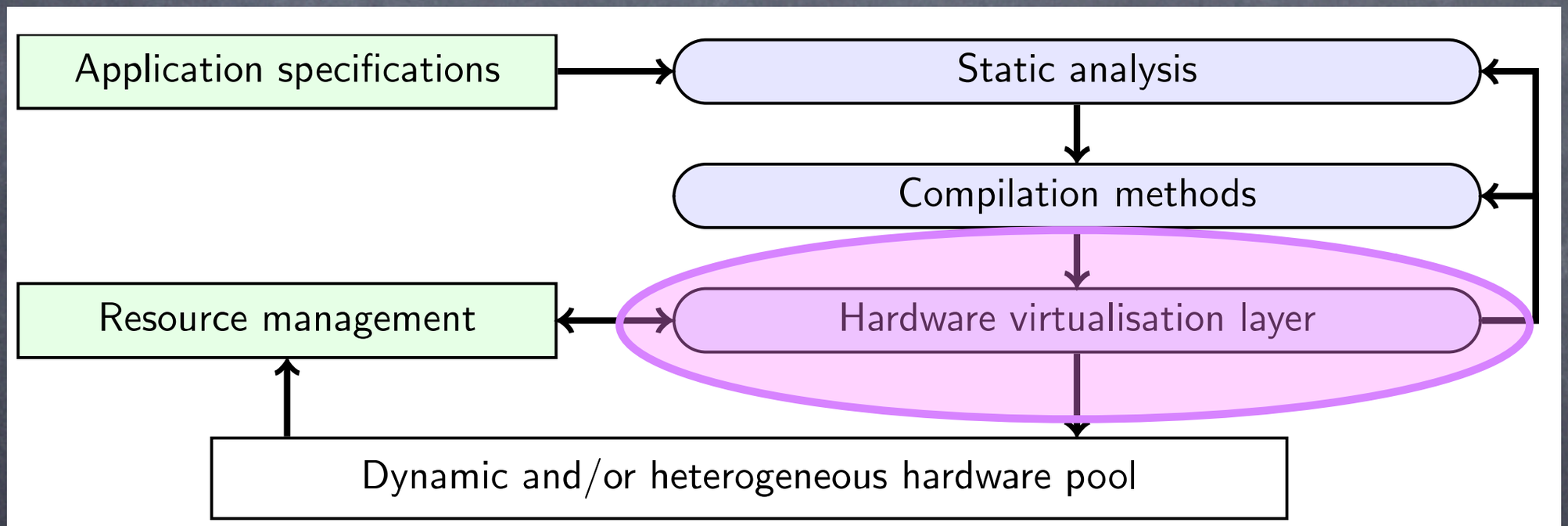
Software engineering ADVANCE

- Program:
  - Capture extra-functional requirements in programs
  - Virtualize hardware and modelize resources in heterogeneous environments
  - Design mapping and feedback mechanisms

apple core

ADVANCE StatArch

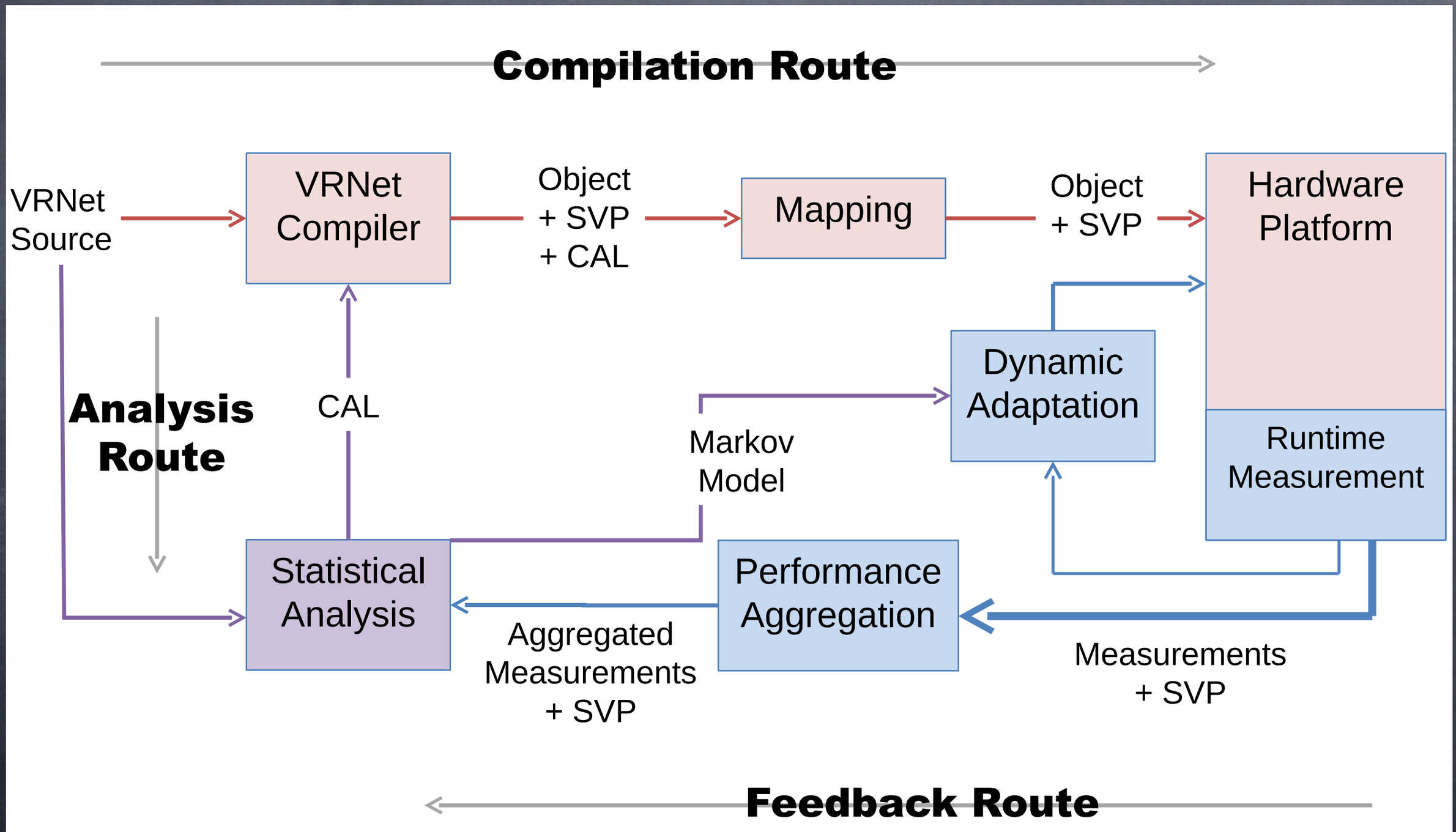
ADVANCE at a glance



apple core

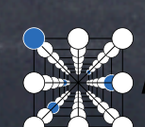
ADVANCE StatArch

ADVANCE interactions



Software engineering issues (ADVANCE)

- Partner technology: S-NET
  (coordination via SISO streams and functional boxes)
- The position of SVP (WIP):
  - Simplify the analysis of requirements of a S-NET network implemented using our protocol
  - Simplify mapping by imposing constraints on scheduling and communication patterns
  - Establish T/L/J models for both internal and external concurrency on heterogeneous cores

Now what?

Further discussion occurs now!