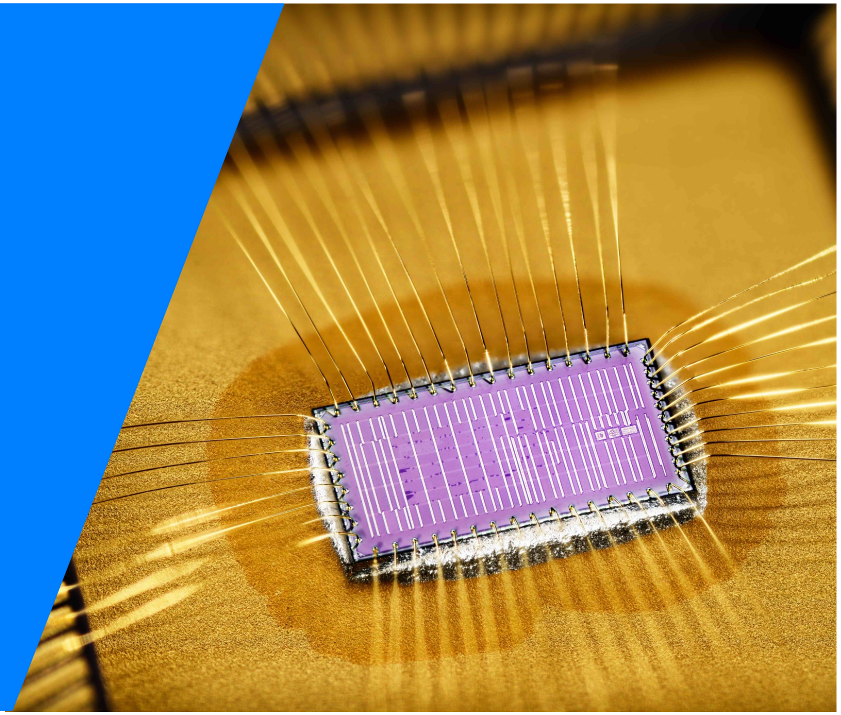
# *Formal resource management for real-time multicore applications*

Kees Goossens
Andrew Nelson, Martijn Koedam,
Mojtaba Haghi, Dip Goswami, Marc Geilen, Twan Basten

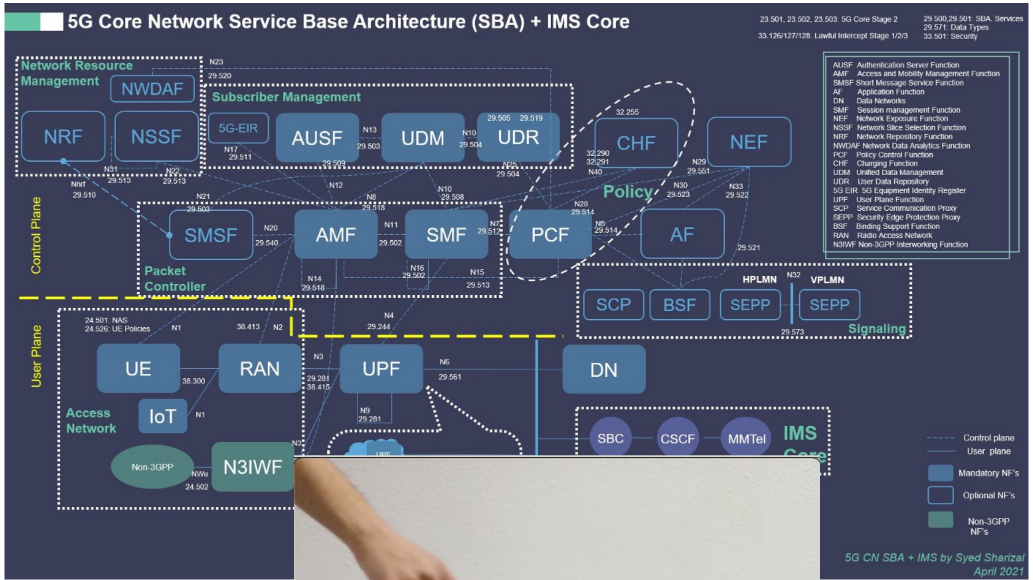


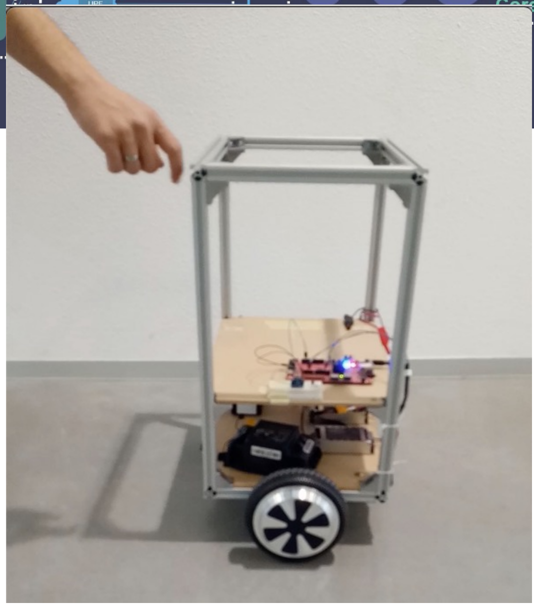
Verintec
Isolating Simplicity

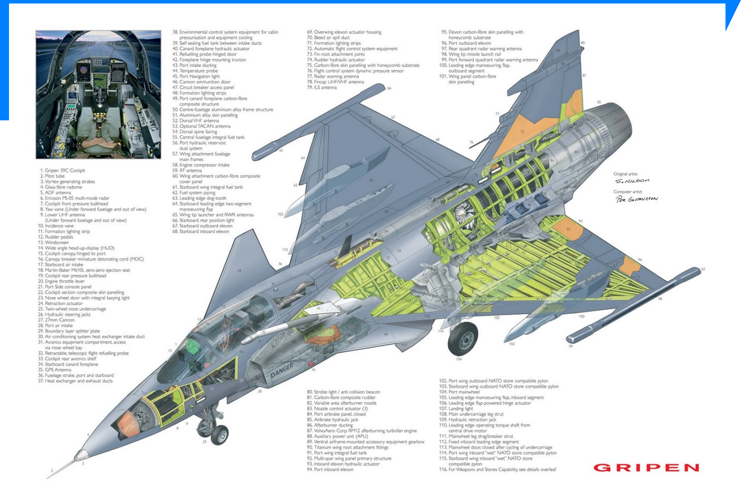
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

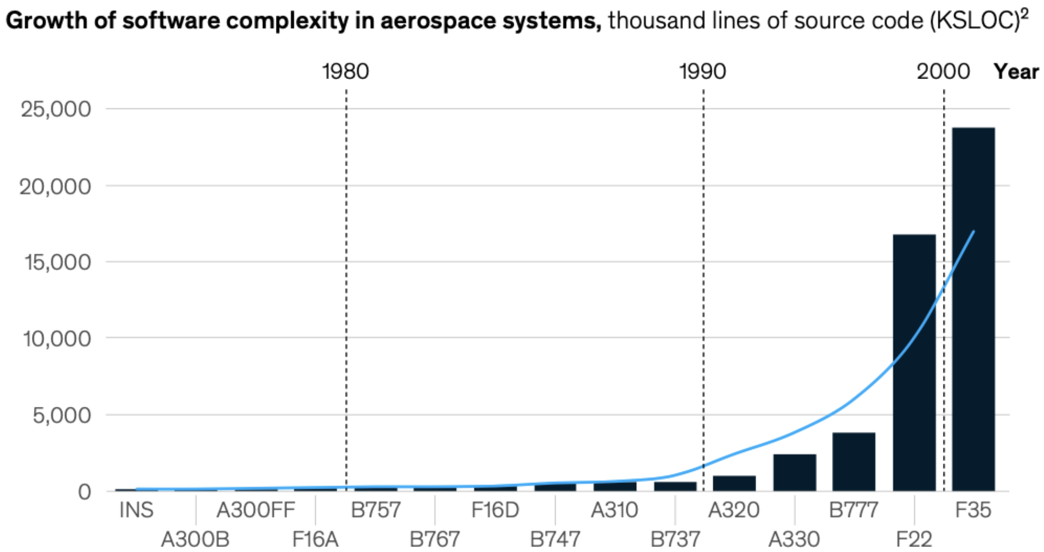
# we're building amazing systems





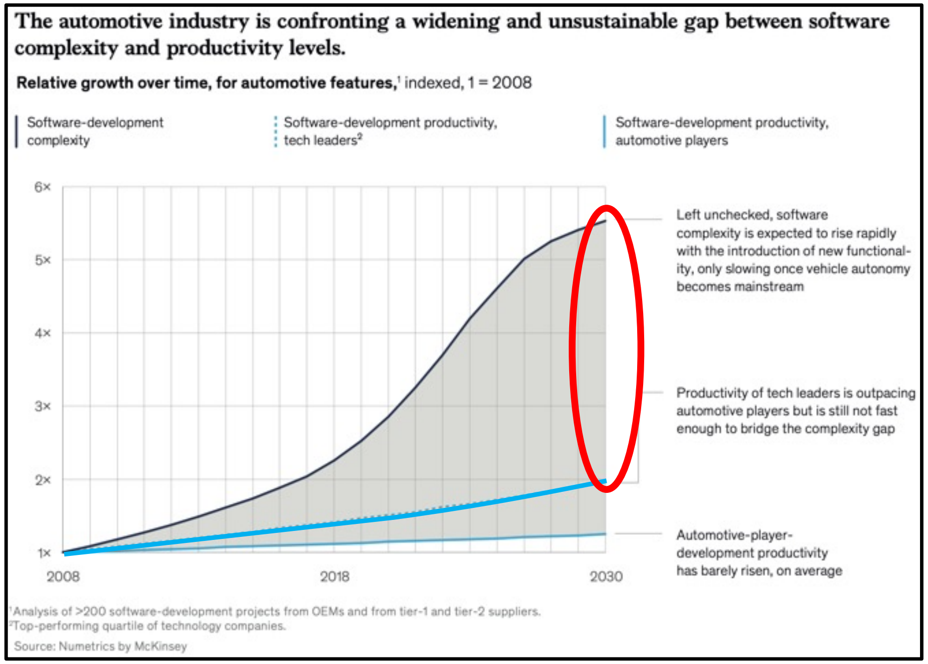


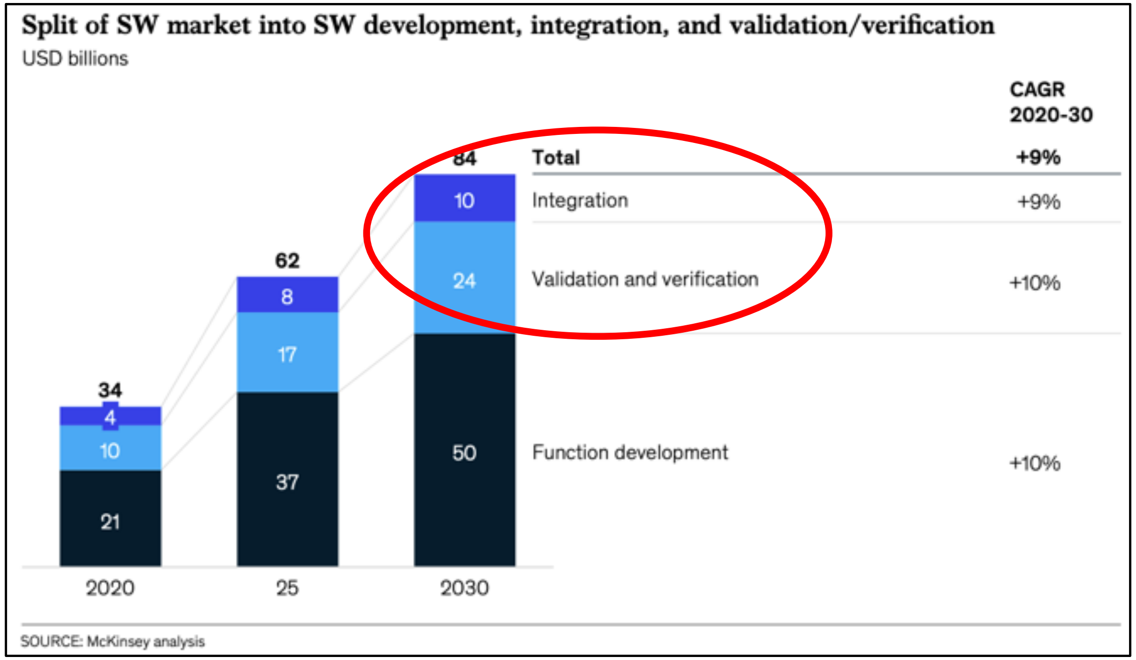




[1]"When code is king: Mastering automotive software excellence," February 17, 2021, McKinsey.com.
[2] Thousands of source lines of code.
Source: Paulo Soares Oliveira Filho, "The growing level of aircraft systems complexity and software investigation," International Society of Air Safety Investigators, 2020, isasi.org; McKinsey's SoftCoster embedded software project database

# but development takes long

- The latest automotive innovations depend on software quality and integration. 30 to 50 percent [of software cost] is commonly dedicated to integration.
  [The case for an end-to-end automotive-software platform, McKinsey, 2020]
  [Cracking the complexity code in embedded systems development, McKinsey 2022]



# (embedded) systems complexity



- complexity slows development, integration, & verification

- single application
  - functionality, scalable, adaptive
  - real time
  - cost:performance trade offs
  - cross-cutting reliability, safety, longevity

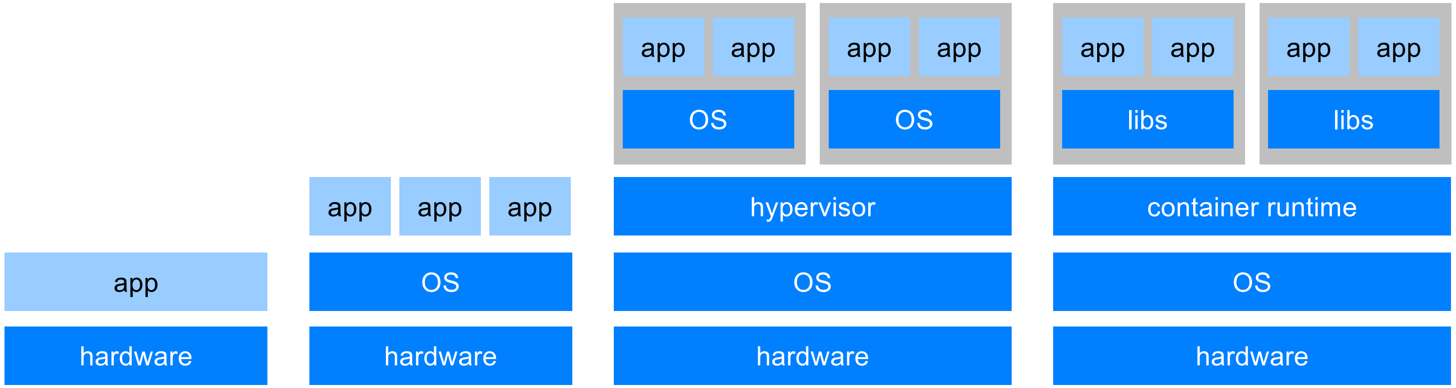
- multiple applications in a single system

# managing multiple applications



- time sharing – operating systems
- virtualisation – hypervisors
- containerisation – runtimes

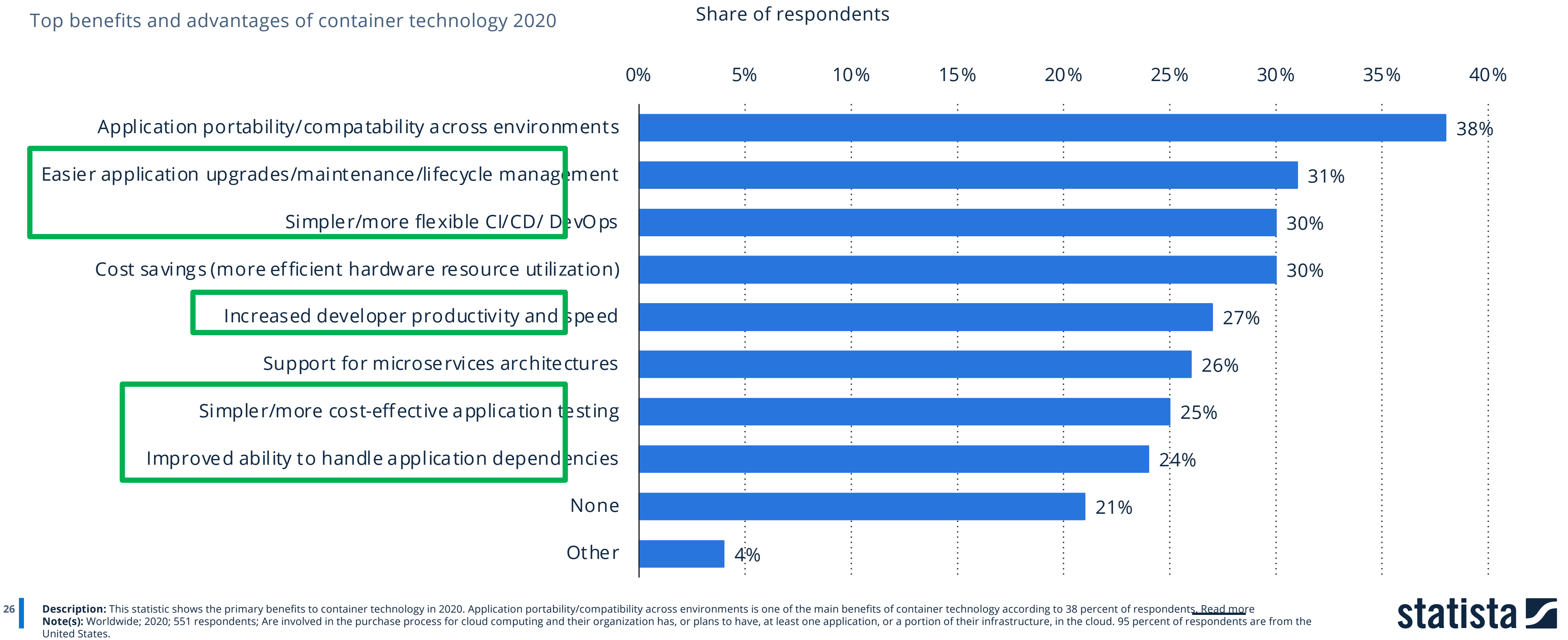
- applications are functionally isolated
- flexible deployment
  - load balancing, migration, etc.



# perceived primary benefits of container technology



- essentially, decoupling applications



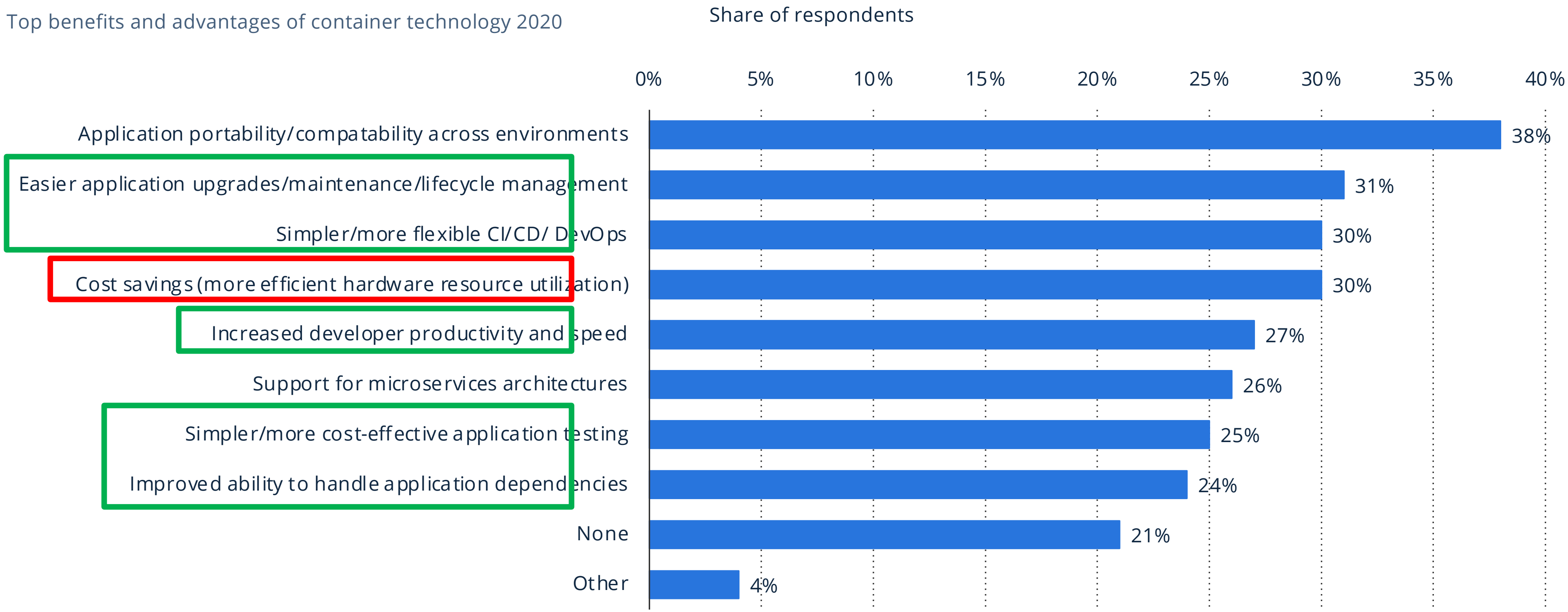

**Description:** This statistic shows the primary benefits to container technology in 2020. Application portability/compatibility across environments is one of the main benefits of container technology according to 38 percent of respondents. Read more
**Note(s):** Worldwide; 2020; 551 respondents; Are involved in the purchase process for cloud computing and their organization has, or plans to have, at least one application, or a portion of their infrastructure, in the cloud. 95 percent of respondents are from the United States.
**Source(s):** IDG Research Services

# perceived primary benefits of container technology



- and statistical multiplexing of resources

Top benefits and advantages of container technology 2020




26 **Description:** This statistic shows the primary benefits to container technology in 2020. Application portability/compatibility across environments is one of the main benefits of container technology according to 38 percent of respondents. Read more
**Note(s):** Worldwide; 2020; 551 respondents; Are involved in the purchase process for cloud computing and their organization has, or plans to have, at least one application, or a portion of their infrastructure, in the cloud. 95 percent of respondents are from the United States.
**Source(s):** IDG Research Services

statista

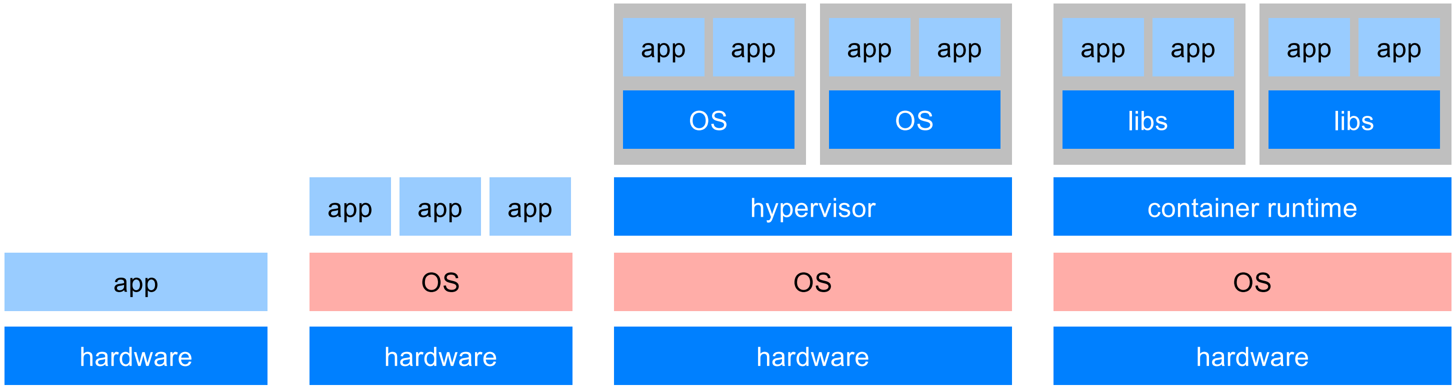
# application performance depends on the allocated resources

- virtualisation & containerisation work well in e.g. hyperscalers with
  - near-infinite resources
  - where load can be shed
  - average performance requirements

- functional correctness
  - spatial isolation
  - limited temporal isolation (focus on average)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

```
$ docker run -it \
    --cpu-rt-runtime=950000 \
    --ulimit rtprio=99 \
    --cap-add=sys_nice \
    debian:jessie
```



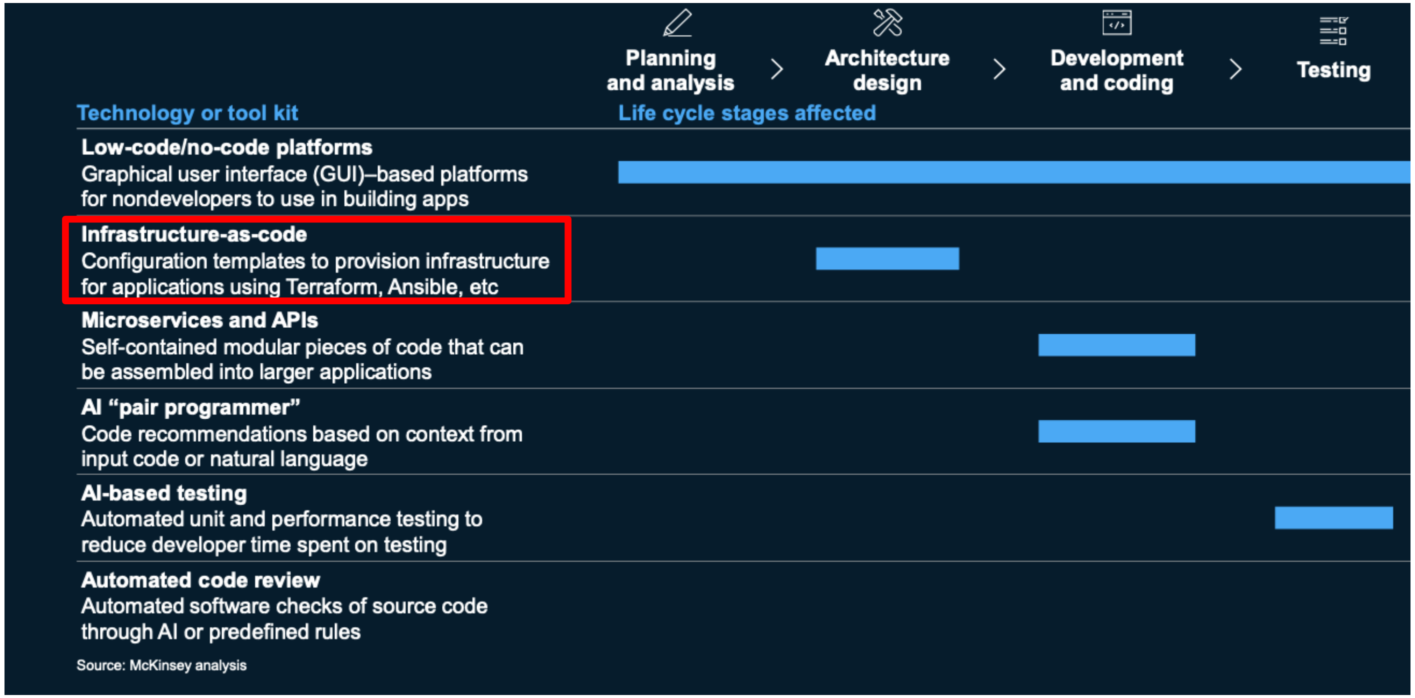
# application performance depends on the allocated resources



- virtualisation & containerisation work well in e.g. hyperscalers with
  - near-infinite resources
  - where load can be shed
  - average performance requirements

- functional correctness
  - spatial isolation
  - limited temporal isolation

- dynamism
  - upgrades
  - changing set of applications
  - Ericsson life-cycle management with ML



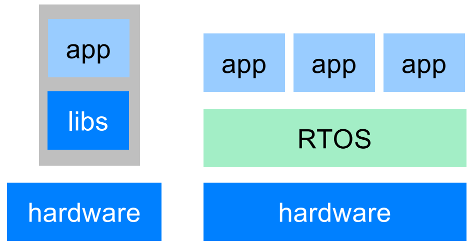

[Next-generation software development, McKinsey Technology Trends Outlook, 2022]

# application performance depends on the allocated resources



- for embedded systems, however,
  - resources are limited
  - performance requirements are richer & stricter, e.g. (soft) real time, jitter, URLL, settling time, MTTF
  - multiple applications can't all be given highest priority

- performance correctness
  - spatial isolation
  - temporal isolation
  - energy/power isolation, I/O




- evolution during long lifetime
  - adaptive applications react to user/environment
  - use cases: changing set of applications
  - upgrades
  - while ensuring performance correctness

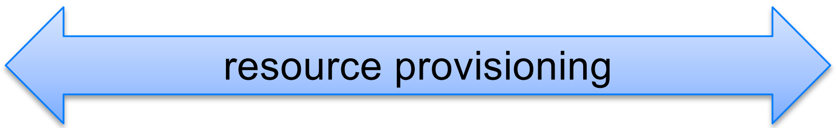
# recap



- system design is hard and takes too long
- general-purpose computing
  - virtualisation & containerisation are good
  - offer only rudimentary Quality of Service
    - resources: not much more than VCPUs
    - application performance?
- embedded systems require strong performance guarantees with limited resources

worse performance guarantees
at higher cost

better performance guarantees
at lower cost

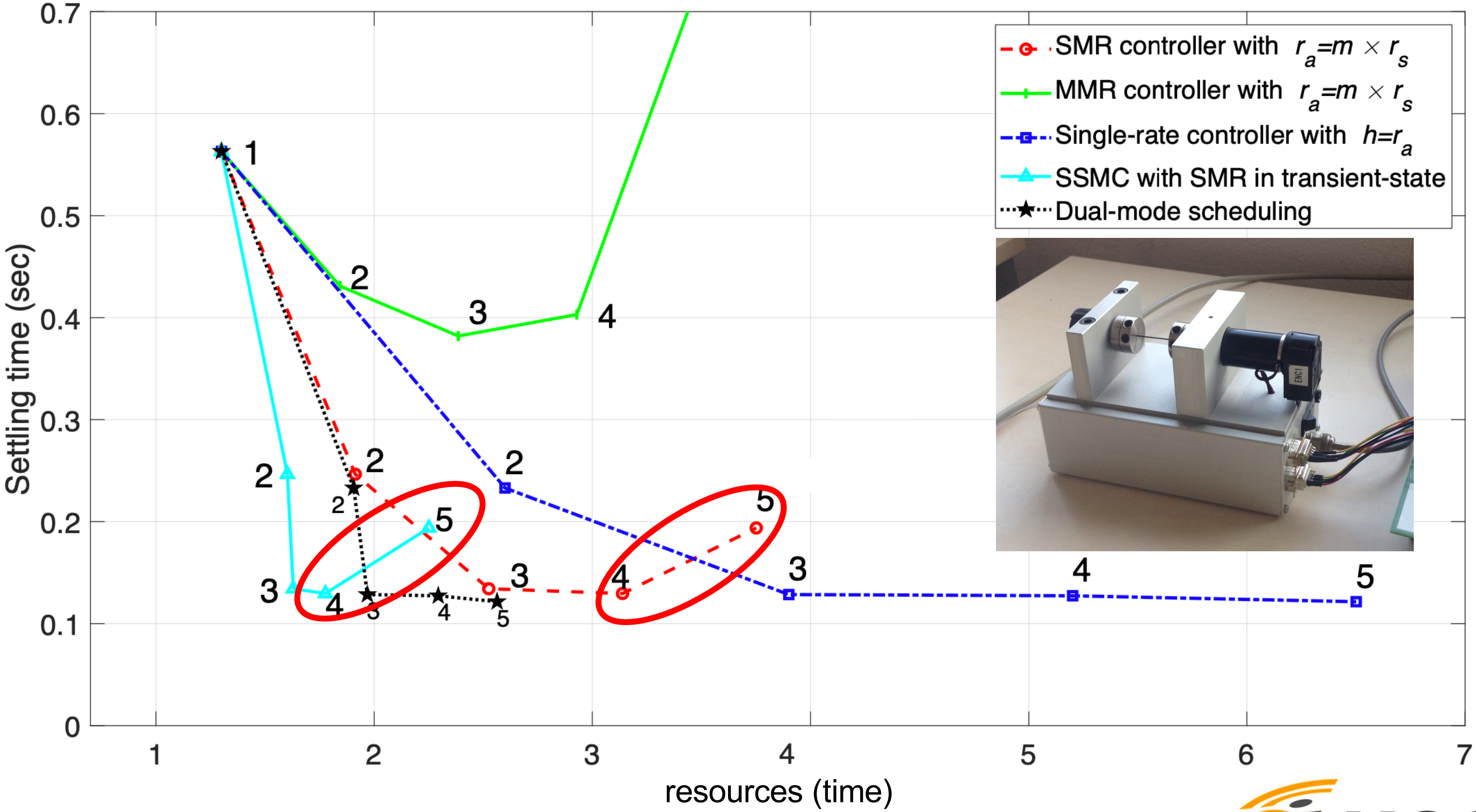



imprecise

precise

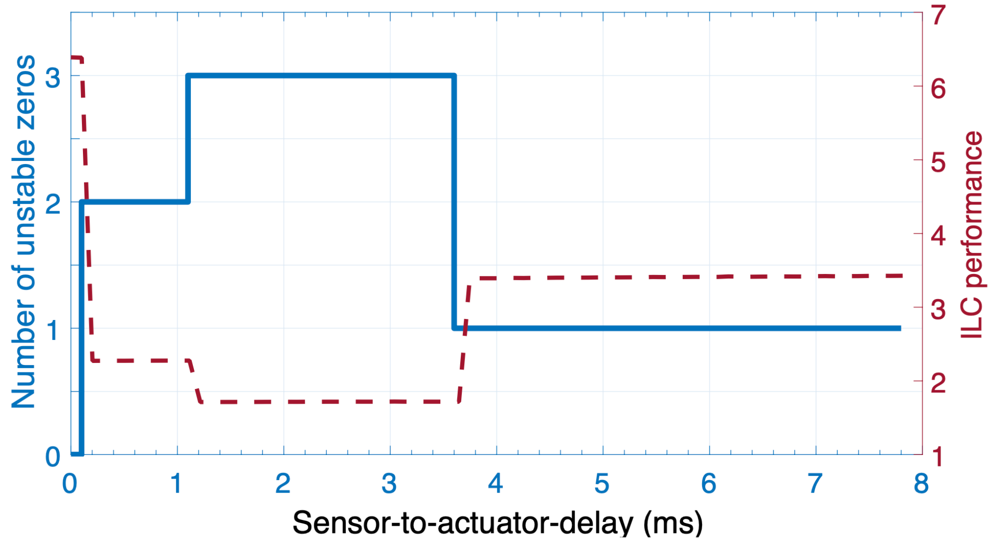
# case study – embedded motion control (cost:performance)





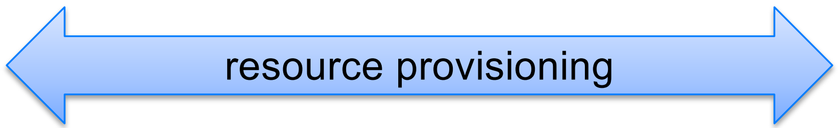
# case study – embedded motion control (non-monotonic)






worse performance guarantees
at higher cost

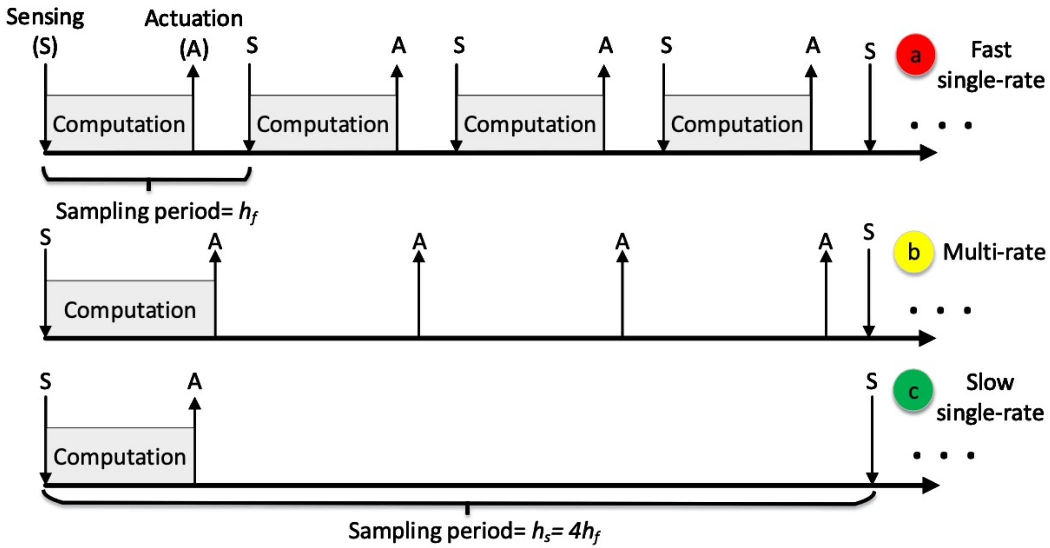
better performance guarantees
at lower cost

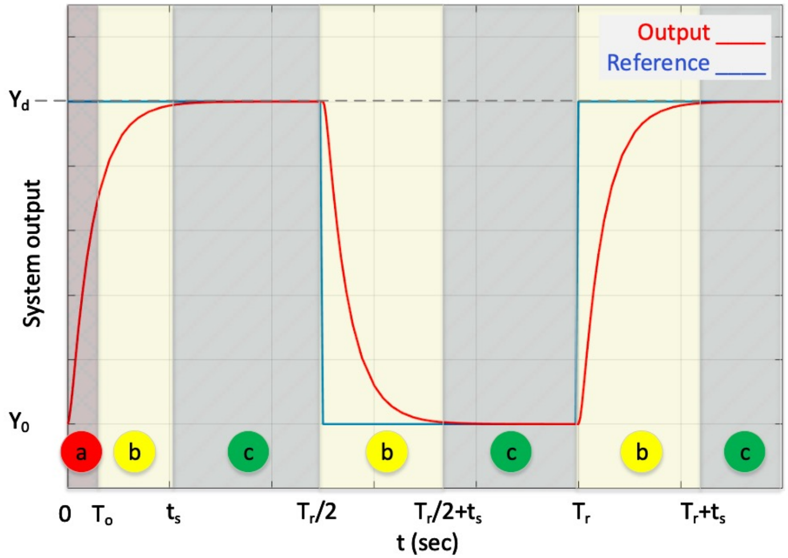



imprecise

precise

# case study – embedded motion control (adaptivity)




worse performance guarantees
at higher cost

same performance guarantees
at lower cost

resource management

imprecise

precise

# CompSOC approach



1. a platform that offers precise resource provisioning

2. formal quality & resource model (QRM) and language (QRML) describing both platform & applications
3. compute Pareto-optimal mapping of applications on platform

4. platform offers precise resource management (deployment)

- QRM & QRML are generic technologies developed in FIT OPTI VIS and further extended in TRANSACT
- CompSOC is just one platform to which it is applied

- QRM is formally defined in *Interface Modeling for Quality and Resource Management,* Martijn Hendriks, Marc Geilen, Kees Goossens, Rob de Jong, Twan Basten, LMCS, 2021, 17(2)
- qrml.org

# CompSOC – 1 – resource provisioning

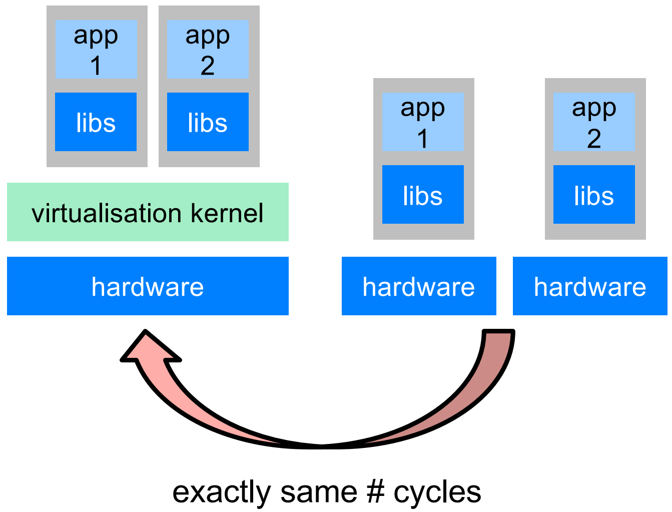


- a hardware/software platform that offers Virtual Execution Platforms (VEP) to applications

- a VEP is a virtualised subset of the platform
  - extremely precise resource provisioning
    - space & time (storage, computation, communication)
  - virtualisation to bare metal
    - i.e. no services like those of an OS or container runtime

→ perfect isolation of applications
  - can be N/S/HRT, adaptive, ...
  - no interference, information leakage, DOS, etc.
  - no need to reverify after integration

- late binding

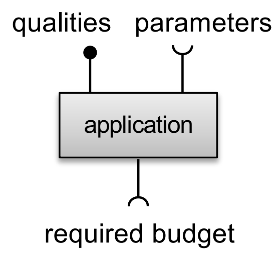


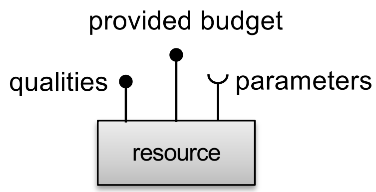
# CompSOC – 2 – quality & resource model



- a hardware/software platform that offers Virtual Execution Platforms (VEP) to applications

- use the QRM model to describe
  - what performance applications offer
    (qualities such as SNR, frame rate, settling time, ...)
  - given what applications require (MIPS, KB, bps, ... budgets)
  - what platform resources offer (MIPS, KB, bps, ... budgets)
  - given what platform resources cost
    (qualities such as energy, power, weight, area, ...)



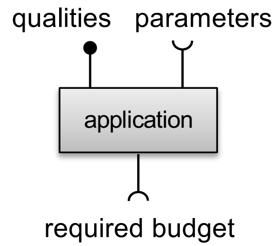



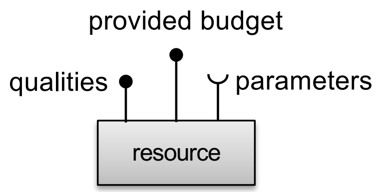
# CompSOC – 3 – quality & resource optimisation



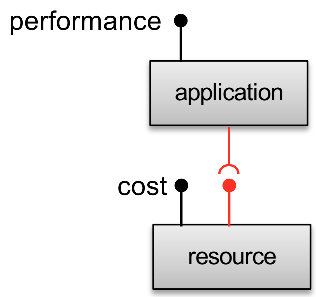
- a hardware/software platform that offers Virtual Execution Platforms (VEP) to applications

- use the QRM model to describe
  - what performance applications offer
    (qualities such as SNR, frame rate, settling time, ...)
  - given what applications require (MIPS, KB, bps, ... budgets)
  - what platform resources offer (MIPS, KB, bps, ... budgets)
  - given what platform resources cost
    (qualities such as energy, power, weight, area, ...)




- broker computes an optimised deployment
  - given a set of platform resources and
  - a set of applications
  - find an application:resource mapping that has a Pareto-optimal cost:performance




- a VEP is the set of virtual resources allocated to an application

# CompSOC – 4 – quality & resource management



- a hardware/software platform that offers Virtual Execution Platforms (VEP) to applications

- a VEP is a virtualised subset of the platform
  - extremely precise resource management
    - cycle-accurate load, start, stop, suspend, resume, snapshot, migrate, etc.
      of the VEP and the application within it

```
./rerun.sh: info: stop all partitions
./rerun.sh: info: (re)loading all partitions (VEPs: 13, partitions: 13/0_1 13/0_2 13/1_1 13/2_1 )
./rerun.sh: reconfiguring TDM schedule at RISC-V clock cycle 0x6c08076e656f (118781740213615 , 27656 / 124675439)
```

# advantages of virtual execution platforms ("resource-precise containers")



- isolate applications → no interference

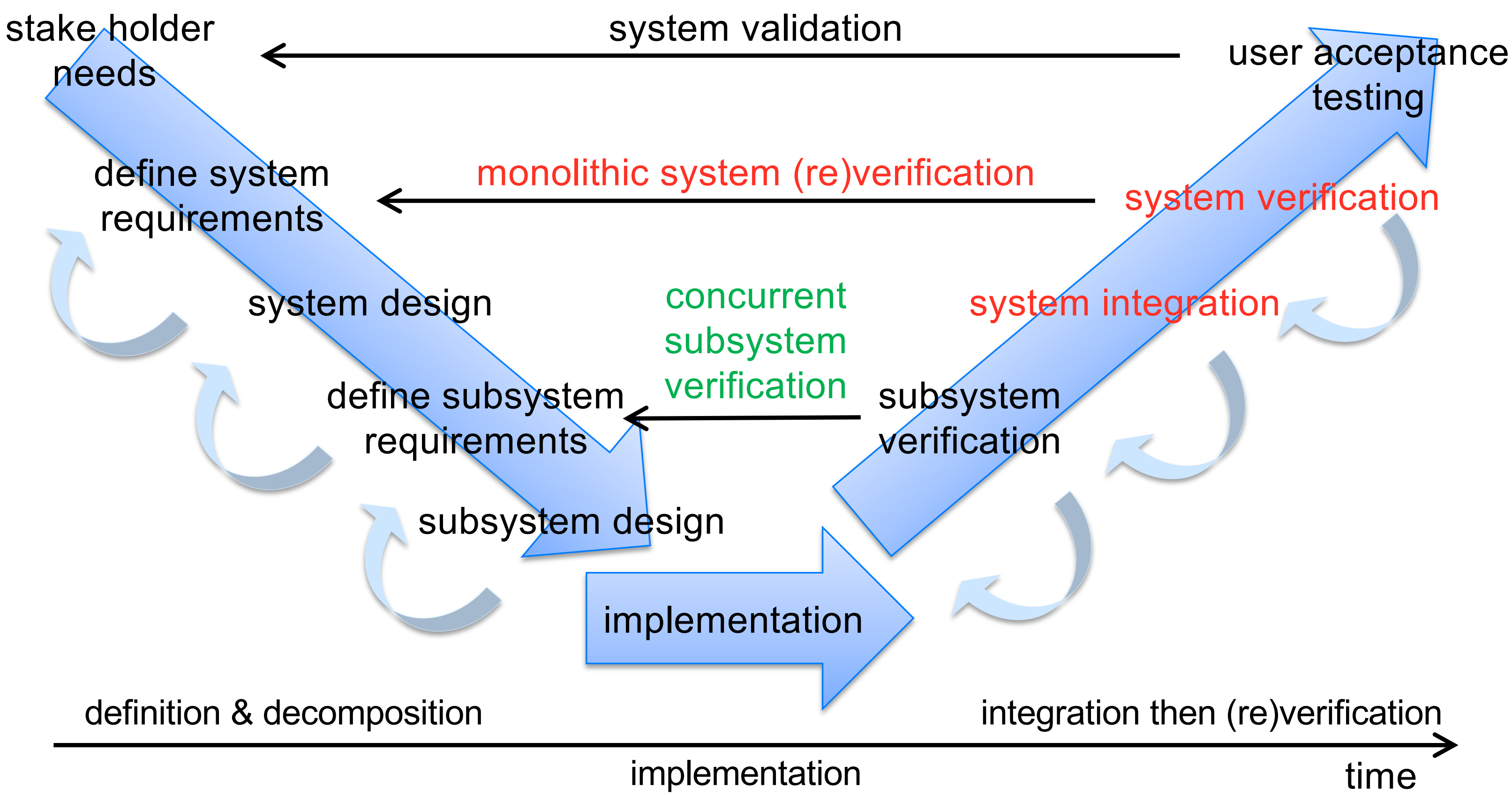
| N! combinations & unbounded interleaving | → | N virtual platforms |
| --- | --- | --- |
| inter-dependent state ($2^S \times 2^S \times 2^S \times 2^S$) | → | independent states ($2^S + 2^S + 2^S + 2^S$) |

- design, debug, verify, run, update each application independently ("composability")
- instead of a monolithic system (performance) verification when applications are ready

- within each VEP, each application can be developed using any appropriate method
  - simulation-based best effort
  - real time with formal model of computation
  - etc.

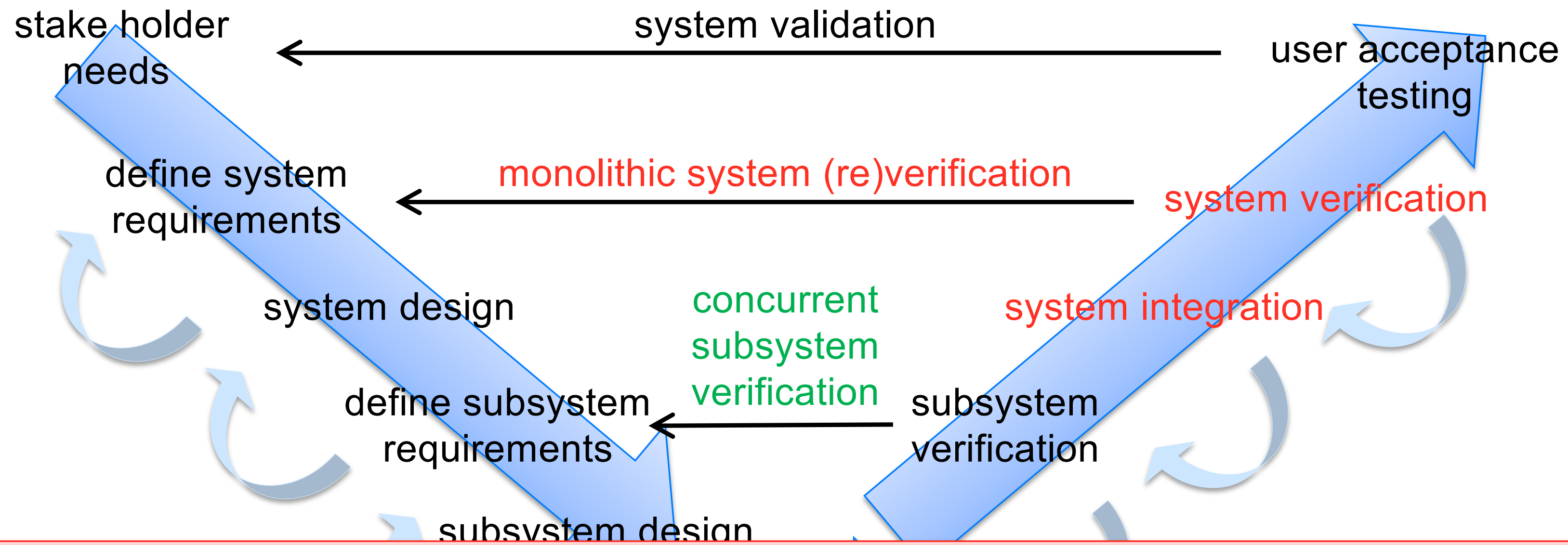
# V model: integration then monolithic system verification





# V model: integration then monolithic system verification



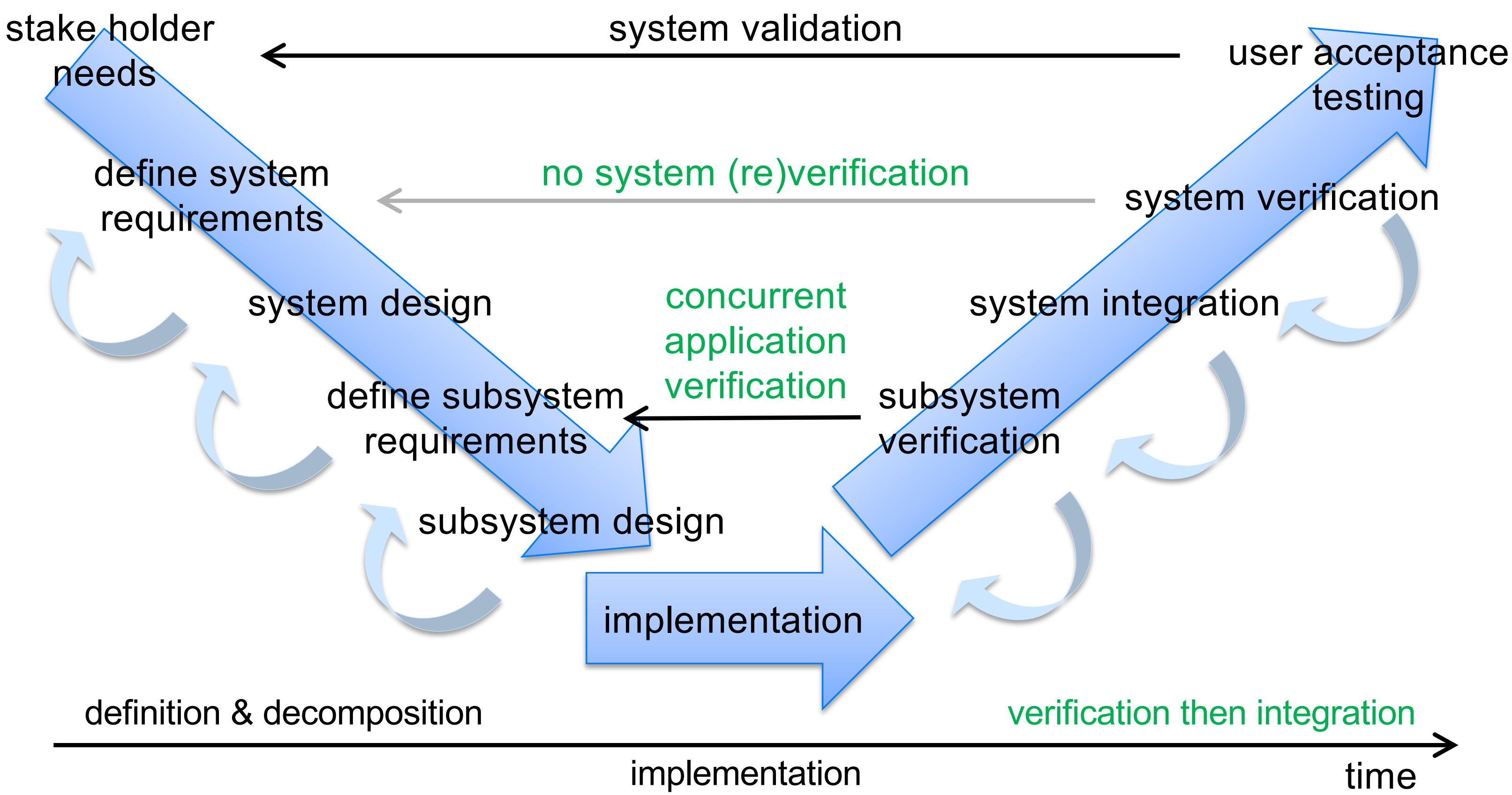



when cars have been hacked, automotive players must provide software updates to fix security issues ... and ensure that software updates will not harm certified safety-relevant systems and are compatible with the vehicles' configuration.
[Automotive software and electronics 2030, McKinsey, 2019]

USAF: "When a modification is incorporated into a system the testing performed can be categorized into two categories. The first is the testing needed to verify and validate the modification being made. The second is to perform regression testing to ensure that the unmodified parts of the system were not unintentionally impacted by the modification effort."
DEPARTMENT OF THE AIR FORCE, Headquarters Air Force Life Cycle Management Center (AFMC), AC-17-01 23 MAR 2017
showing compliance to a set of criteria found in MIL-HDBK-516C, Section 15, *Computer Systems and Software (CS&S)*, which is used in the United States Air Force (USAF) airworthiness certification process.

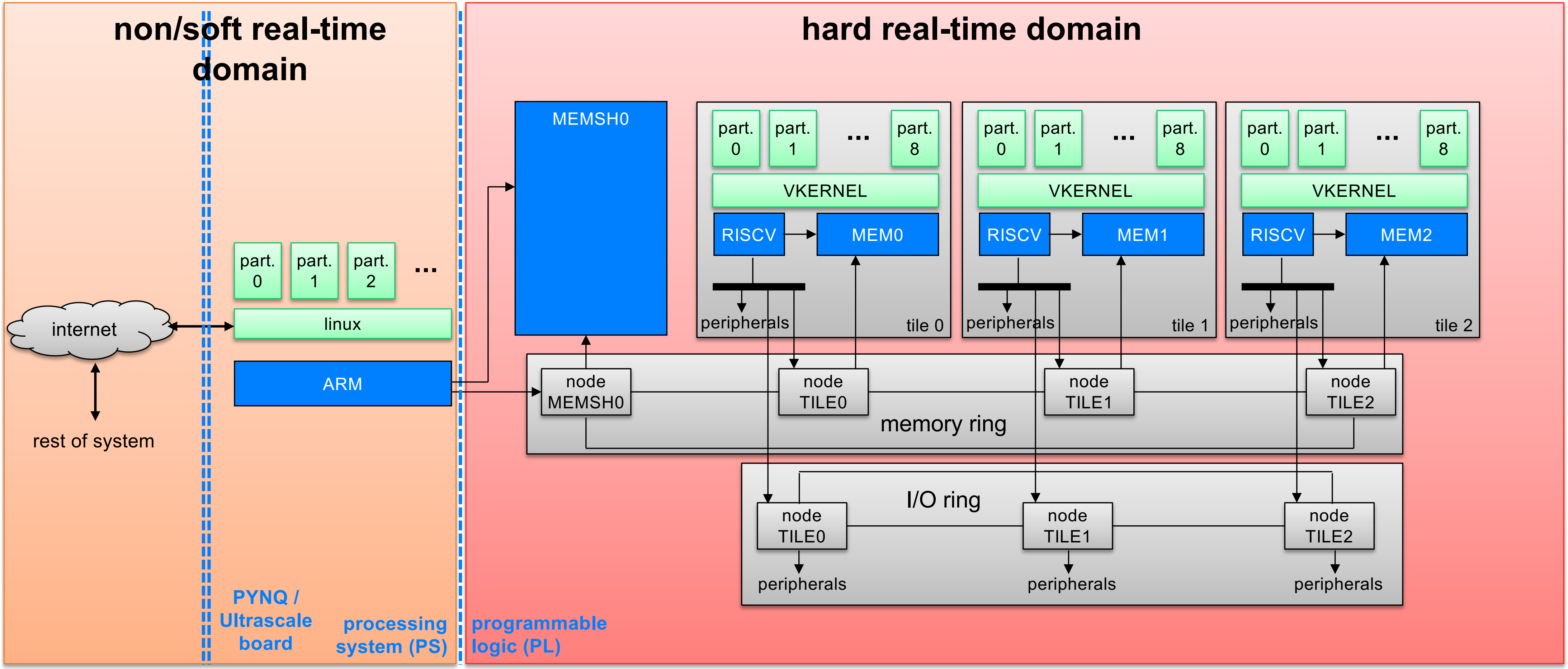
# composability in the V model: verification before integration





# CompSOC template



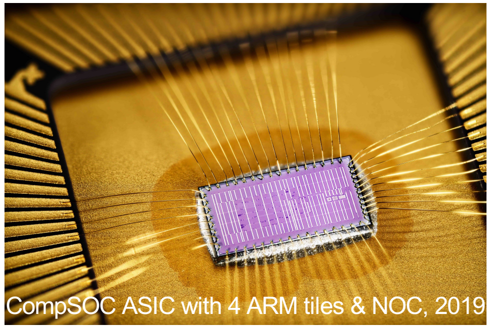


# CompSOC template

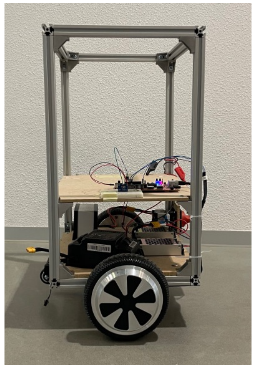


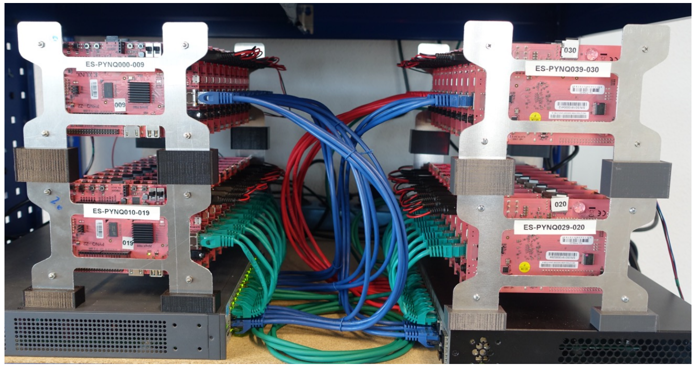
- configurable # tiles with
  - RISC-V
  - local scratchpad, no caches
  - local peripherals (timers, etc.)
  - virtualisation kernel with cycle-accurate TDM
- configurable # memory tiles
- token rings for
  - memories
  - global, shared peripherals
- MMUs for space partitioning
- memory regions can be selectively shared
  - within an application
  - between applications
- synchronous

- predictable → well-defined precise WCET& WCRT
- composable → space + time partitioned
  → stateless application switching
- virtualised → identical # cycles with(out) virtualisation



CompSOC ASIC with 4 ARM tiles & NOC, 2019





# formal resource management: budgets



- to be predictable, resource usage must be budgeted & enforced
- composable budgets must be non work conserving (think: TDM)
- best-effort applications also have budgets, but with less good service

- a budget is binary: you either get it or you don't
- when you get it, it is guaranteed until you relinquish it

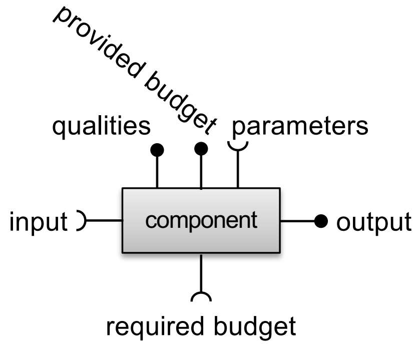
- reserving a budget on a resource results in a virtual resource
- budgets & (virtual) resources are hierarchical
  - platform > tile > processor > DMEM

- a virtual execution platform is a set of virtual resources

# formal resource management: components



- component
  - set of configurations
    - parameter (determines the configuration)
    - quality (cost, performance)
    - output
    - input
    - provided budget
    - required budget
    - initial state (required to instantiate component)



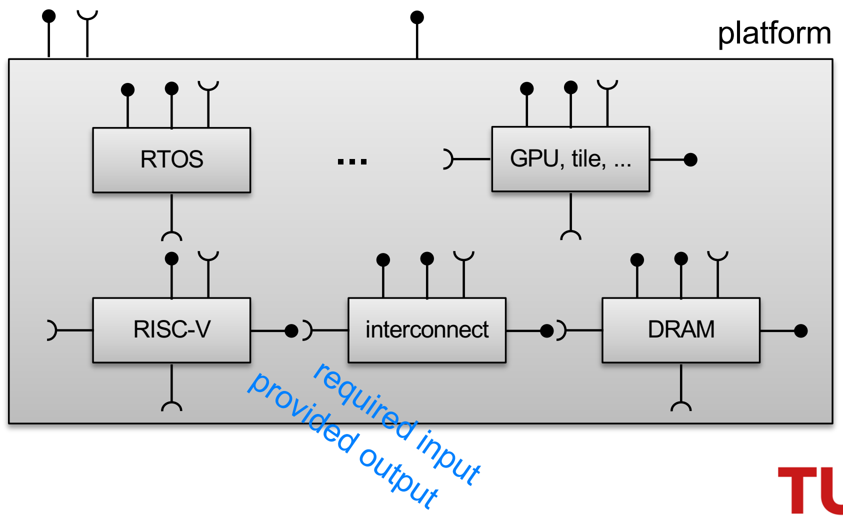

- configurations capture
  - different modes (e.g. single video, picture in picture; sleep vs. burst mode)
  - different implementations & different mappings (at different cost:performance points)
  - Pareto set

# component composition

- components can be composed
  - horizontally: I/O
  - vertically: provided – required budgets (services)
  - hierarchically

- application
- virtual execution platform
- execution platform

(hierarchical) set of components



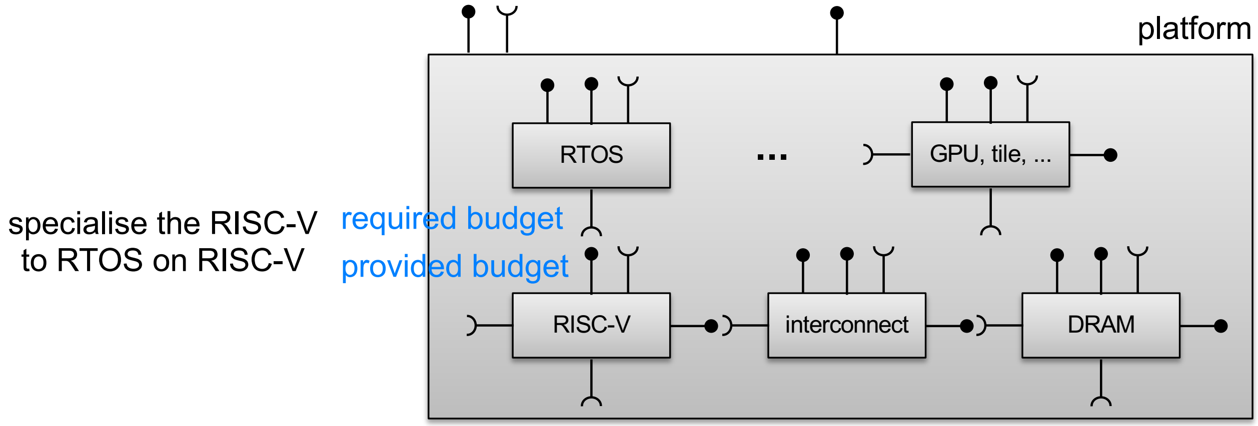
# component composition



- components can be composed
  - horizontally: I/O
  - vertically: provided – required budgets (services)
  - hierarchically

- application
- virtual execution platform
- execution platform

} (hierarchical) set of components



# component composition

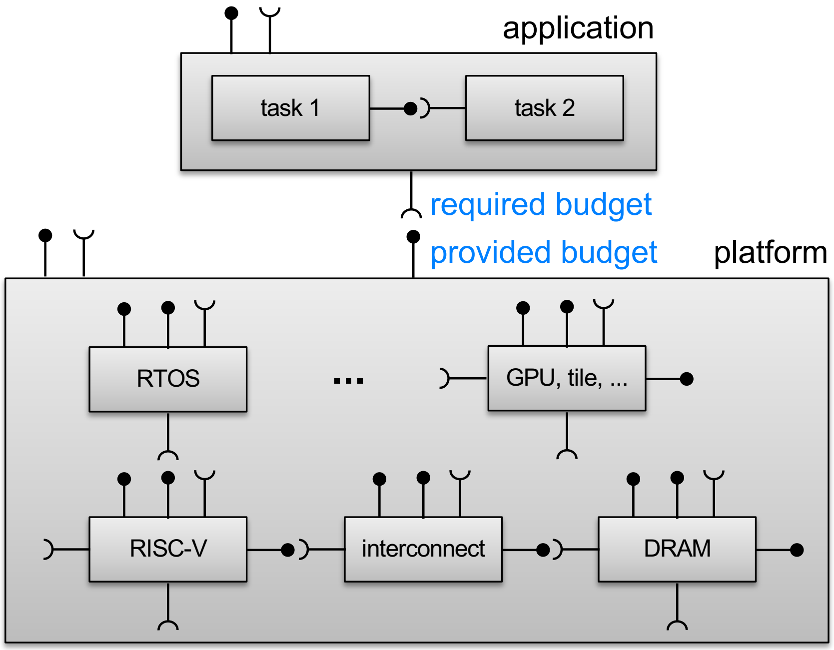


- components can be composed
  - horizontally: I/O
  - vertically: provided – required budgets (services)
  - hierarchically

- application
- virtual execution platform
- execution platform

} (hierarchical) set of components

running a single application on a platform



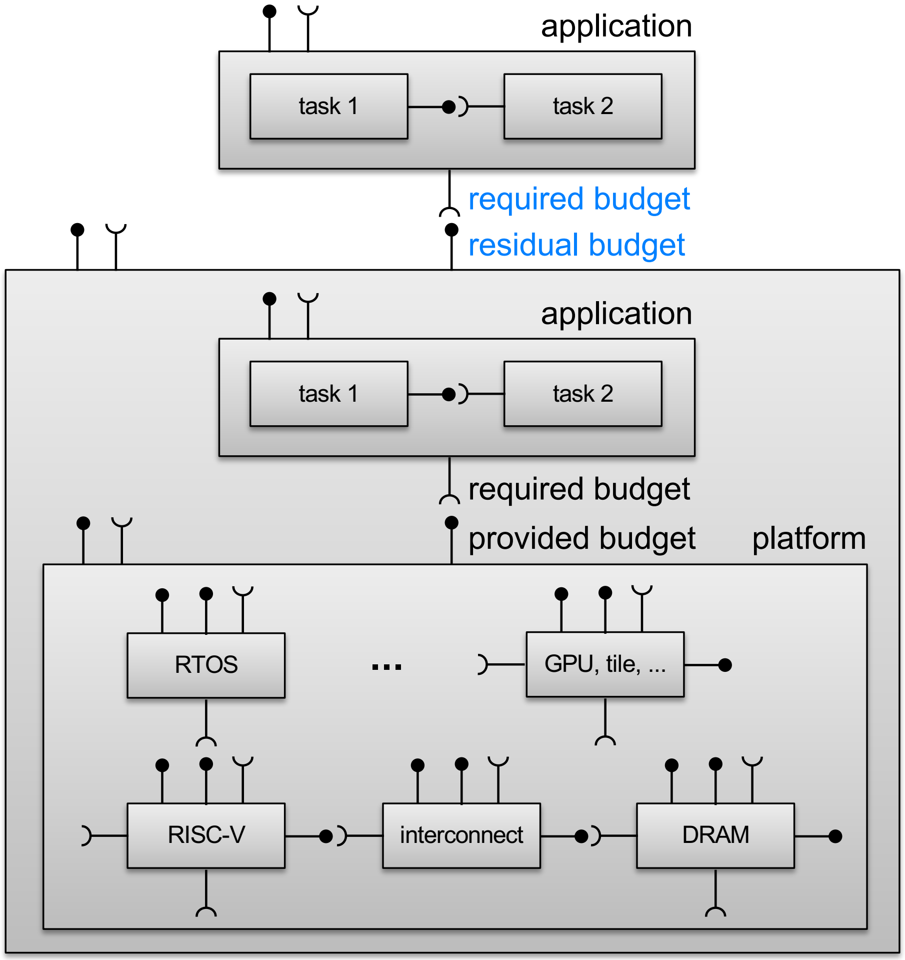
# component composition



- components can be composed
  - horizontally: I/O
  - vertically: provided – required budgets (services)
  - hierarchically

- application
- virtual execution platform
- execution platform

(hierarchical) set of components



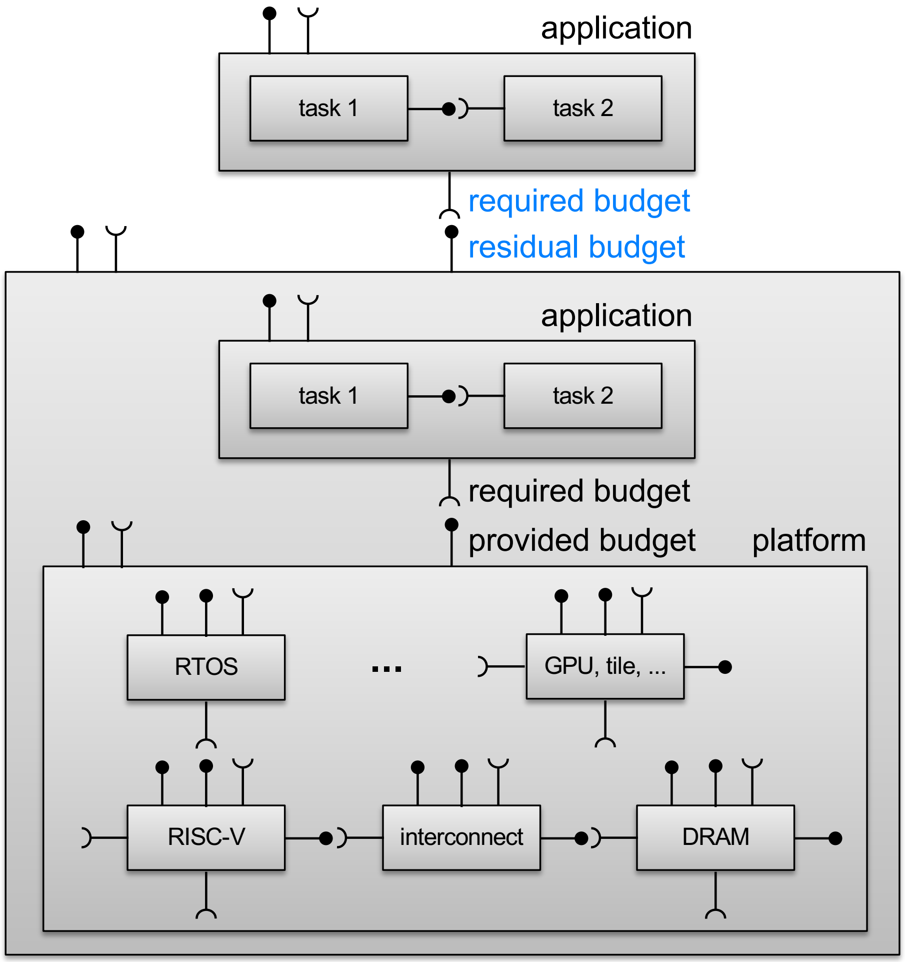
# component composition & optimisation



- composition requires

1. budget matching
   - output >= input
   - provided >= required

2. satisfying user parameter & quality constraints
   - e.g. high resolution, max. 10 Watt



# component composition & optimisation

- composition requires

1. budget matching
   – output >= input
   – provided >= required

2. satisfying user paramete
   – e.g. high resolution, m

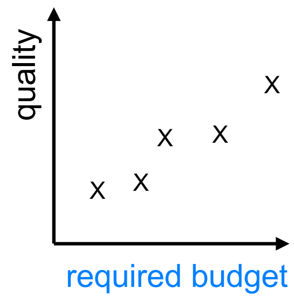

```
"components":
[
  {
    "id" : "Task1",
    "configurations":
    [{
      "inputs":[{"raw_frames" : "30Hz"}, ...],
      "outputs":["processed_frames" : "30Hz"}, ...],
      "parameters":[{"resolution" : "720p"}, ...],
      "qualities":[{"framerate" : 30}, ...],
      "required_budget":
      {
        "TILE": { "RISCV":
                  [{
                    "unit" : "cycles",
                    "type" : "average_rate",
                    "value" : 100K
                  },... // other services from RISCV
                ], ... // other resources from TILE
                }, ... // other resources besides TILE
      },
      "initial_state":[{"IDMEM" : ".../task1.hex"}, ...]
    }, ... // other application configurations
    ]}, ... // other components
],
"compositions":
[
  "App1 = Task1 => Task2",
  ...
]
```


3. Pareto optimisation
   – pick best configuration with high resolution, max. 10 Watt
   – use Z3 SMT solver at design time (or in cloud @ run time) or run-time embedded heuristics

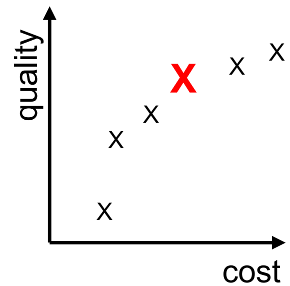
   → declarative description of the best VEP(s)

4. deploy the VEP

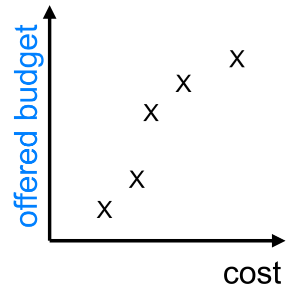
**Infrastructure-as-code**
Configuration templates to provision infrastructure for applications using Terraform, Ansible, etc




application

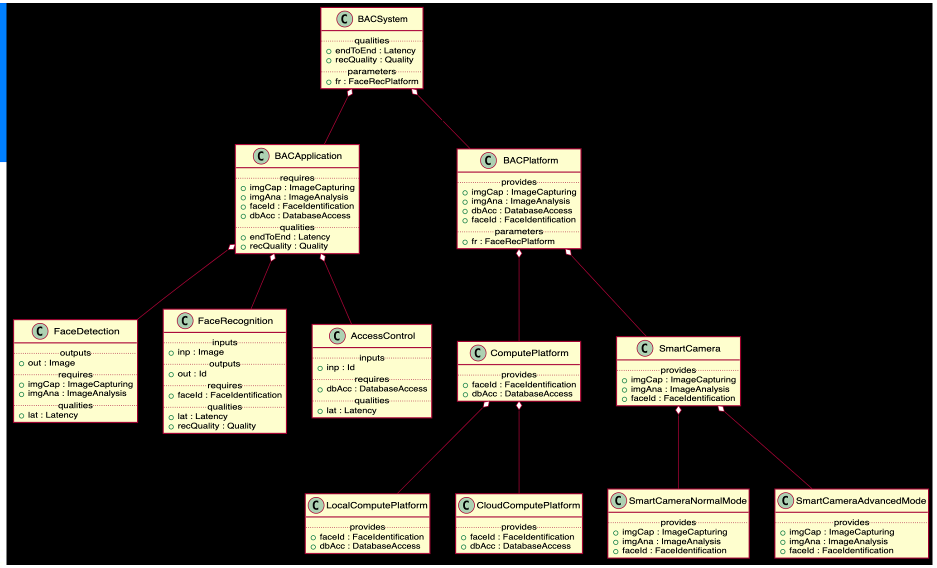



optimise
(application X execution platform)




execution
platform

# QRML example

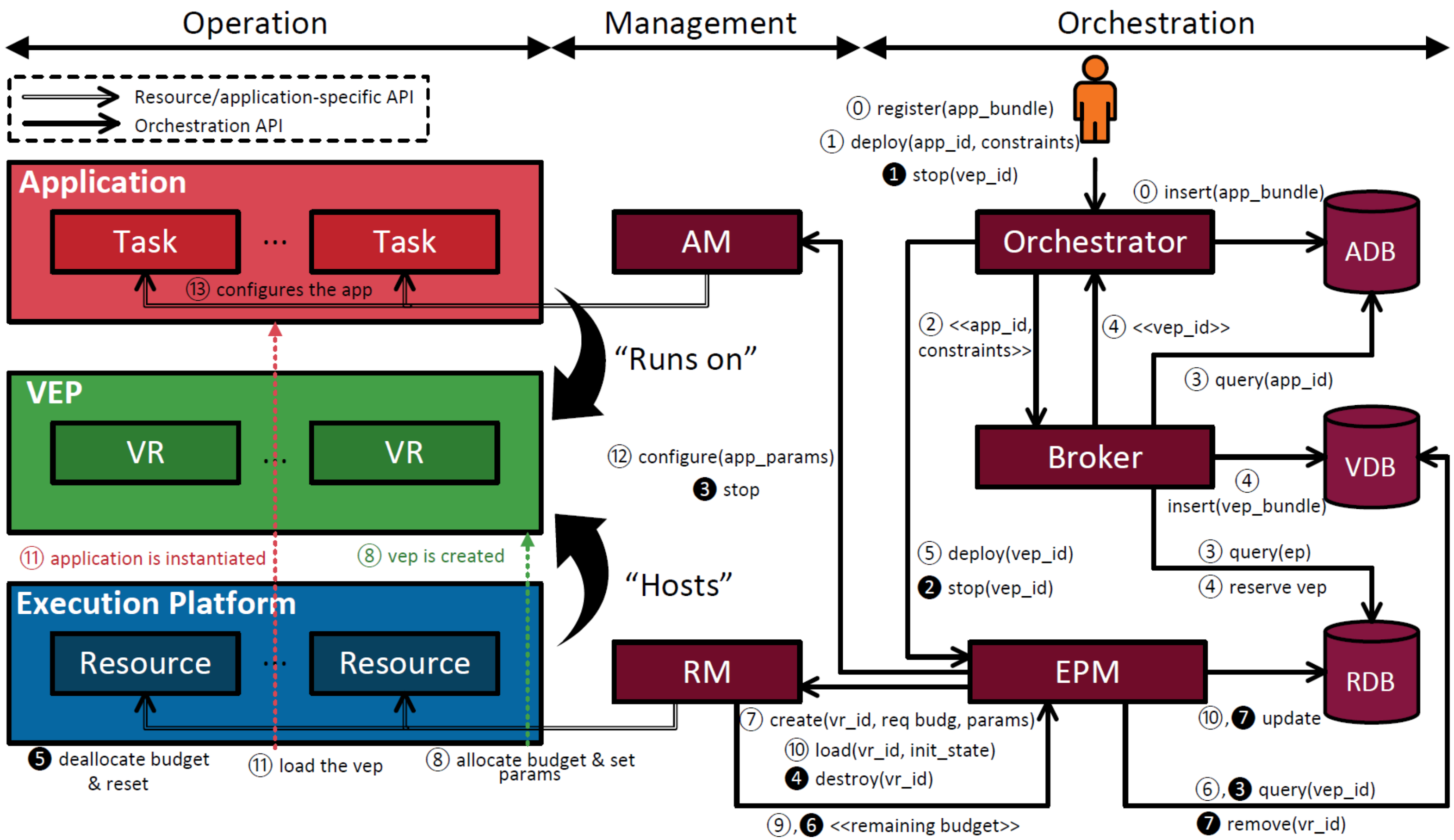



```
budget Bw integer

budget FrameRate integer

budget Computation integer

channel Video {
    hres : integer ordered by =
    vres : integer ordered by =
    rate : integer ordered by =
} ordered by a<=b if a.hres <= b.hres & a.vres <= b.vres & a.rate <= b.rate

budget Scaling {
    segs : integer
    comp : Computation
} ordered by element-wise

budget Scalers {
    streams : integer
    scaling : Scaling
}
```

```
component HWscaler {
    provides Scalers p_sc { streams = 4; scaling.comp = 300; scaling.segs = 32 }
}

component SWscaler {
    provides Computation p_cmp { p_cmp = 100 }
}

component HWorSWscaler {
    contains HWscaler hs or SWscaler ss

    provides Scalers p_sc

    constraint p_sc = hs.p_sc
    constraint p_sc = [top, ss.p_cmp, top]
}
```

# resource management framework





# challenge



- we have

1. a platform that offers precise resource provisioning
2. formal quality & resource model (QRM) and language (QRML) describing both platform & applications
3. formal Pareto-optimal mapping of applications on platform
4. platform offers precise resource management (deployment)

- traditionally, we have clear separate phases
  - design all applications
  - map applications together on platform
  - load & run
  - verify performance
  - repeat