Building Adaptable Software for Resilience

Milind Kulkarni

CRISP Seminar, 9/27/2018

CRISP

PURDUE
UNIVERSITY

what do we mean by resilience?

- When people think **software resilience**, they often think of specific issues:
  - **Fault tolerance:** The ability to tolerate failures during application execution (hardware failure, failure of other software components, etc.)
    - Maybe the most common sense. Microsoft defines resiliency as “the ability of a system to gracefully handle and recover from failures”
  - **Security:** The ability to withstand malicious users or environment

a broader view

resilience, noun: an ability to recover from or easily adjust to **misfortune** or **change**

a broader view

resilience, noun: an ability to recover from or easily adjust to **misfortune** or **change**

software resilience, *noun*: an ability for software to **adapt** to **change**

software resilience as adaptability

- Software should be able to adapt to different circumstances
  - Hardware platform not the same as expected: *failing* hardware, *removing* hardware, *adding* hardware, *changing* hardware
  - Usage scenario not the same as expected: input is *erroneous*, input is *malicious*, input is *larger* or *smaller* than expected

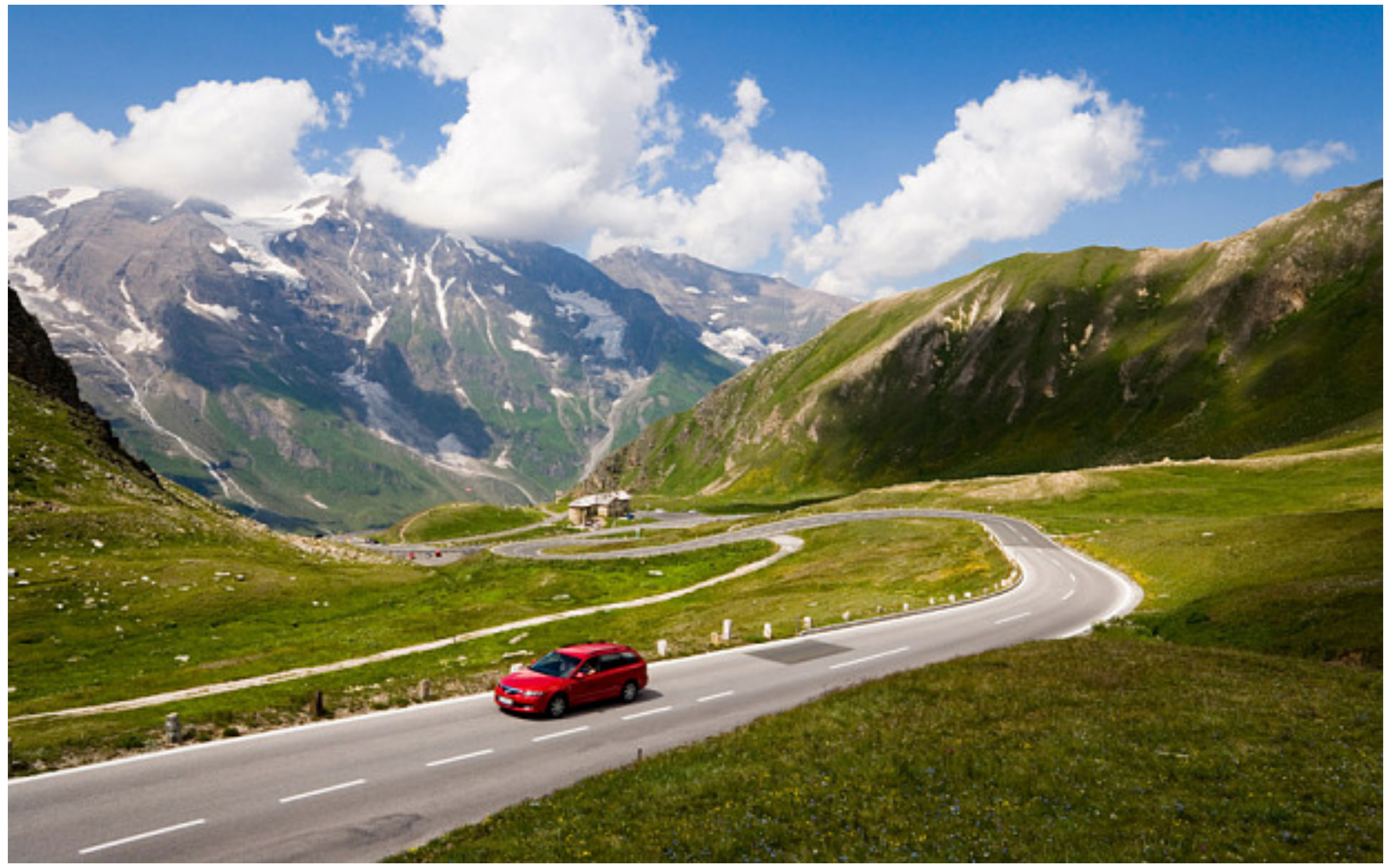
- **Goal**: software written one time, but can adapt to a wide variety of different scenarios

adaptation scenarios

- **Hardware availability can change dramatically due to failure**

adaptation scenarios

- **Hardware availability can change dramatically due to failure**

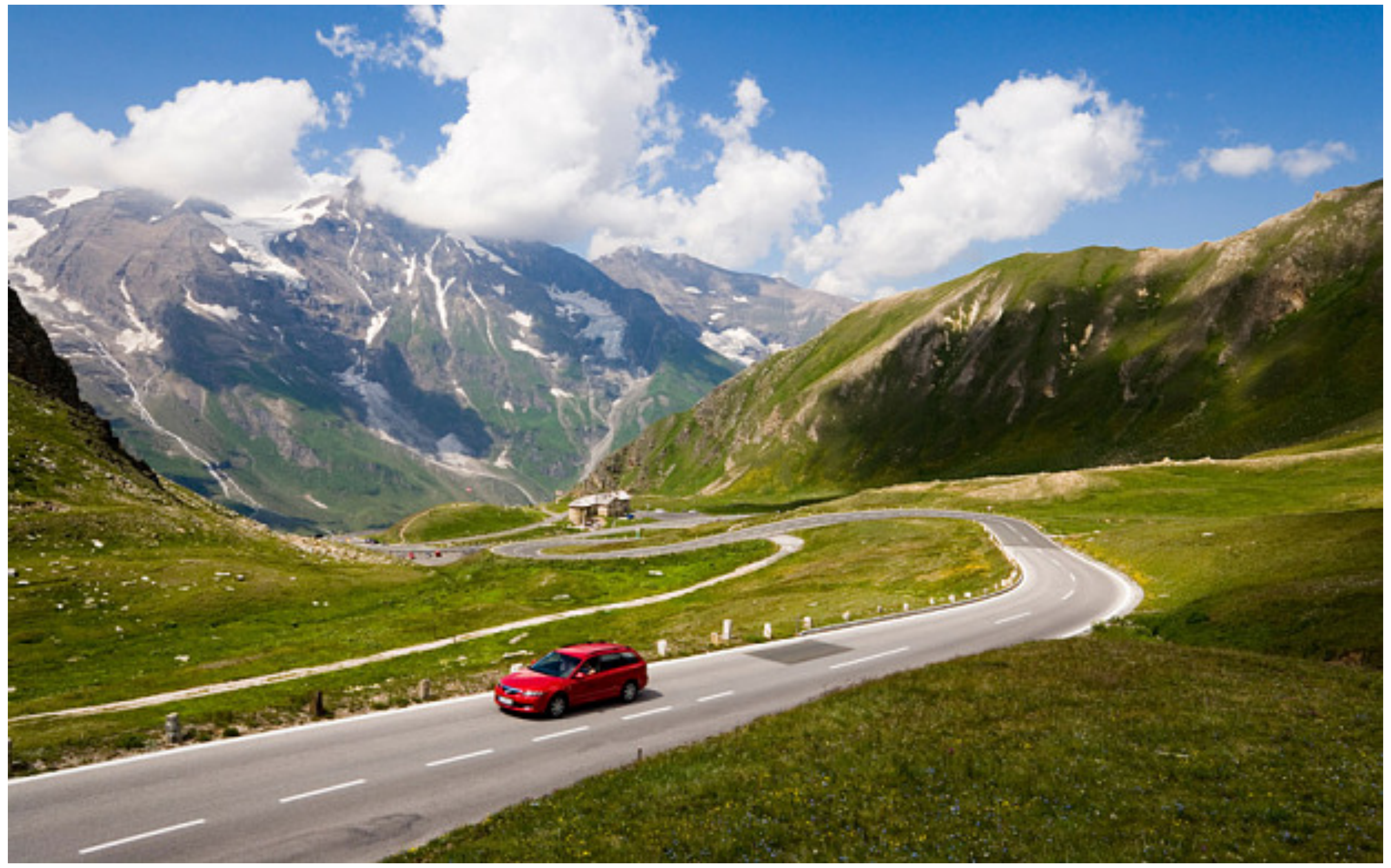


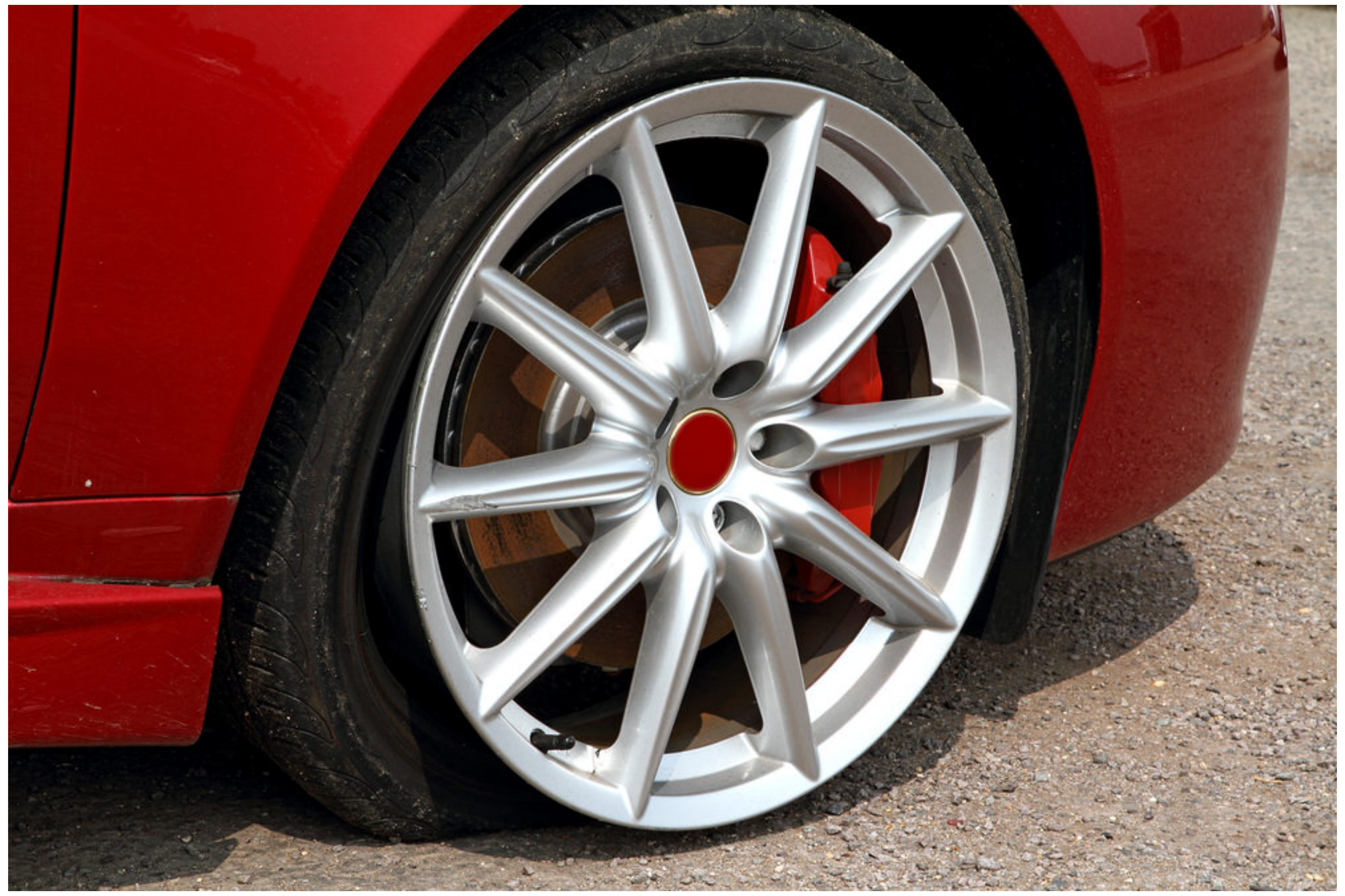
CRISP

PURDUE UNIVERSITY

adaptation scenarios

- **Hardware availability can change dramatically due to failure**





adaptation scenarios

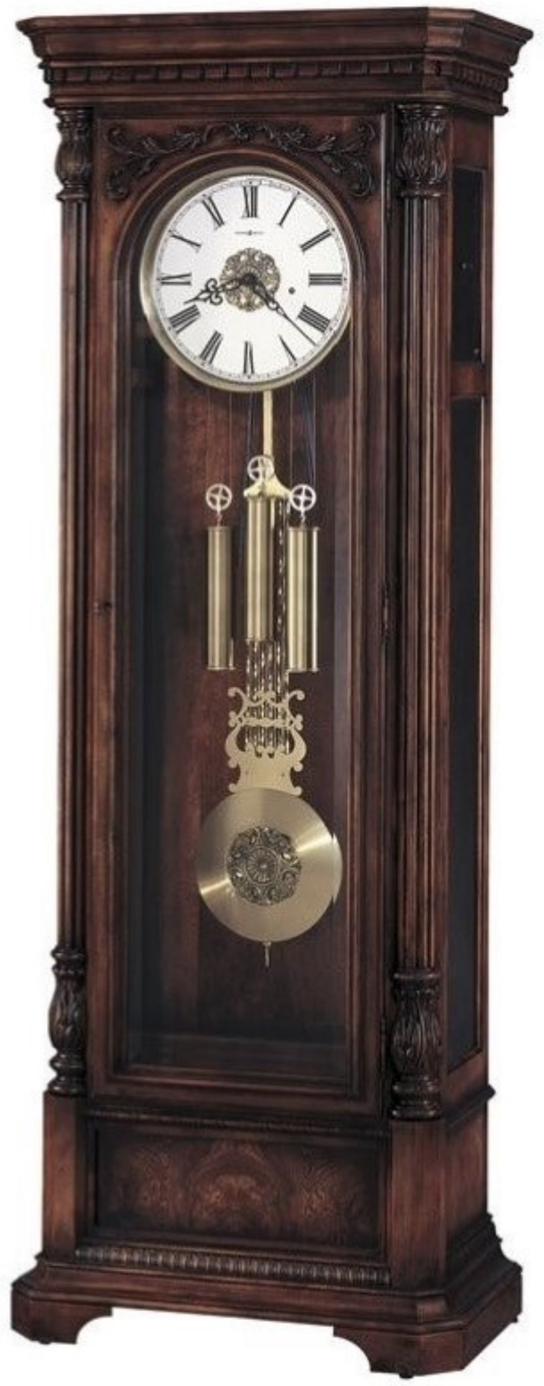
- **Hardware availability can change dramatically due to failure**
    - This is “normal” fault tolerance: how can software react to unexpected changes in the operating environment such as the failure of hardware
    - A wide range of techniques (too many to list)
    - But these are not the only adaptation scenarios to consider!

parameter tuning

- Key hardware parameters vary across systems

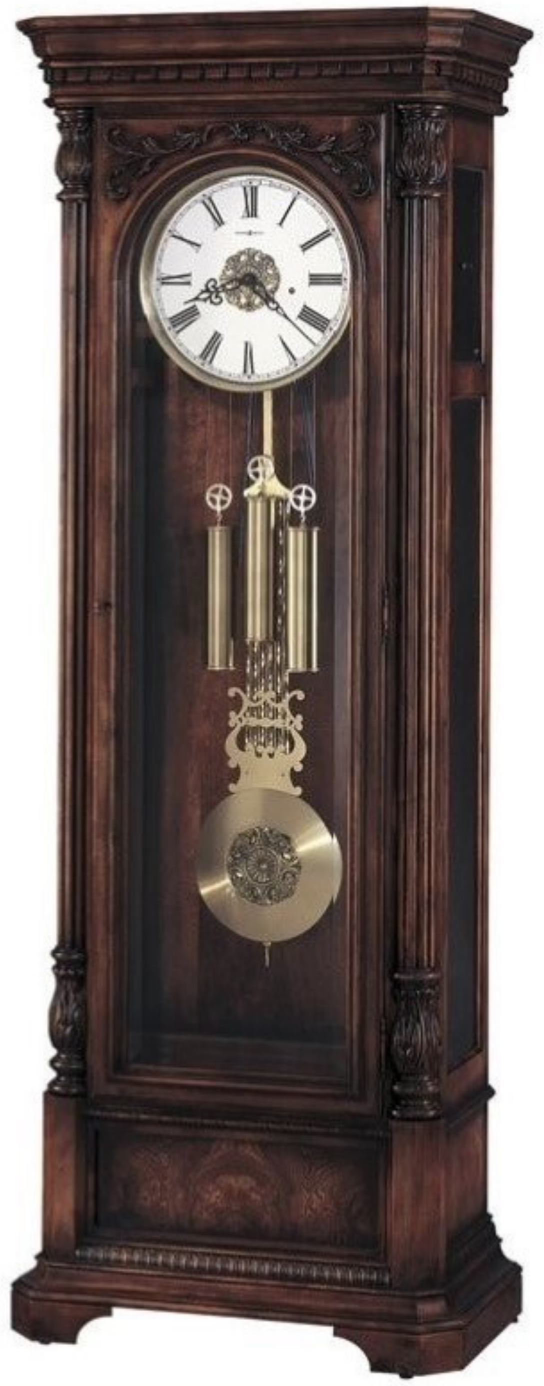
parameter tuning

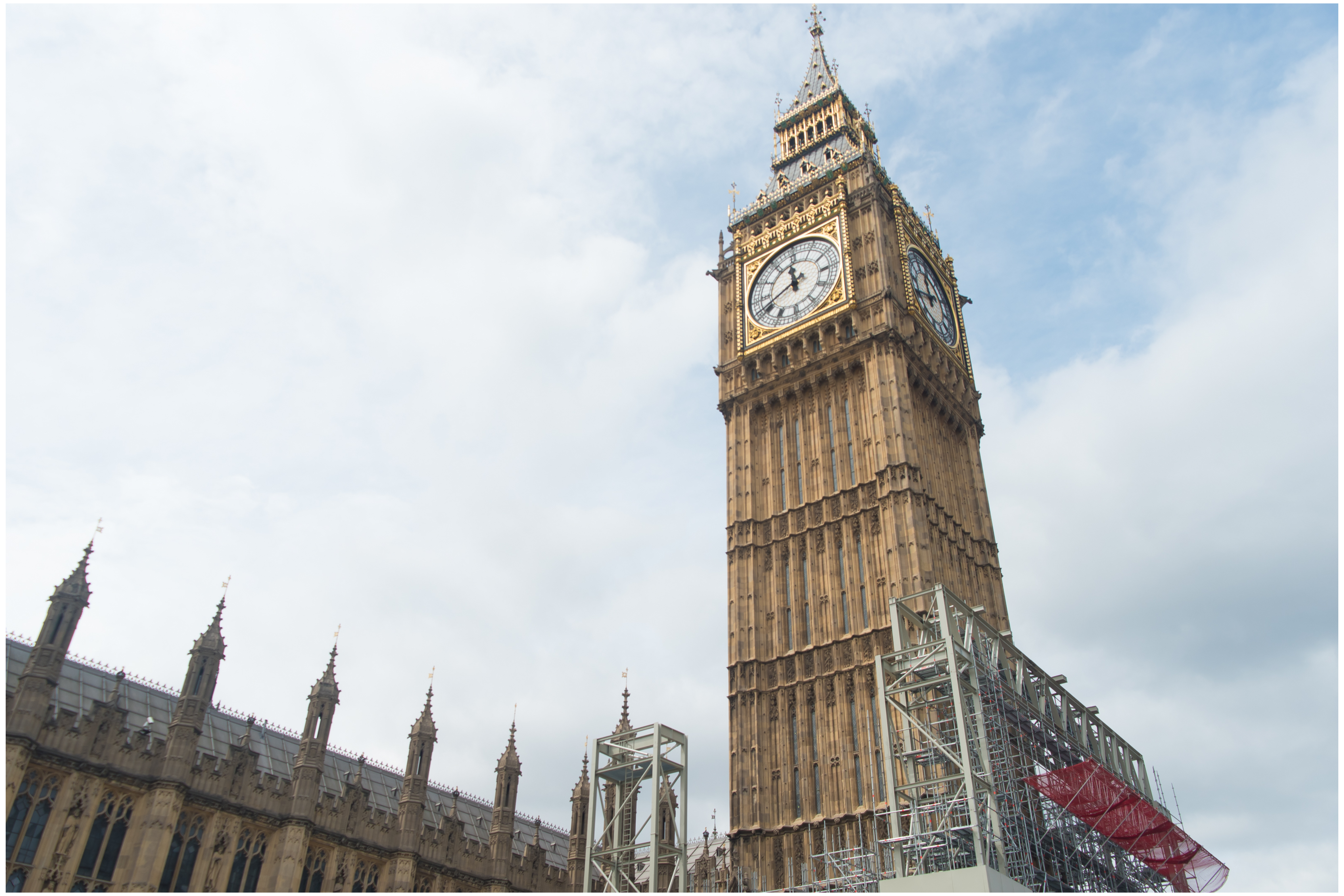
- **Key hardware parameters vary across systems**



parameter tuning

- Key hardware parameters vary across systems





parameter tuning

- **Key hardware parameters vary across systems**
  - Same architecture, but different specific properties (more cores, less memory, etc.)
  - Often can be handled by changing parameters in software (number of threads, parameters that govern cache usage, etc.)
    - Basic algorithms/implementations usually do not change
  - Often done at compile time or install time (hardware parameters like this do not usually change at run time)
    - Not always possible — some scenarios require different parameters for different inputs, so you need to change at runtime
  - **One challenge**: how do you determine what to set a parameter to?

parameter tuning

- **Key hardware parameters vary across systems**
  - Same architecture, but different specific properties (more cores, less memory, etc.)
  - Often c f threads, param
    - Bas
  - Often c his do not usually
    - Not always possible — some scenarios require different parameters for different inputs, so you need to change at runtime
  - **One challenge**: how do you determine what to set a parameter to?

Loop tiling is a classic compiler technique to restructure a loop so its working set fits in cache. The tile size must be tuned for different cache sizes

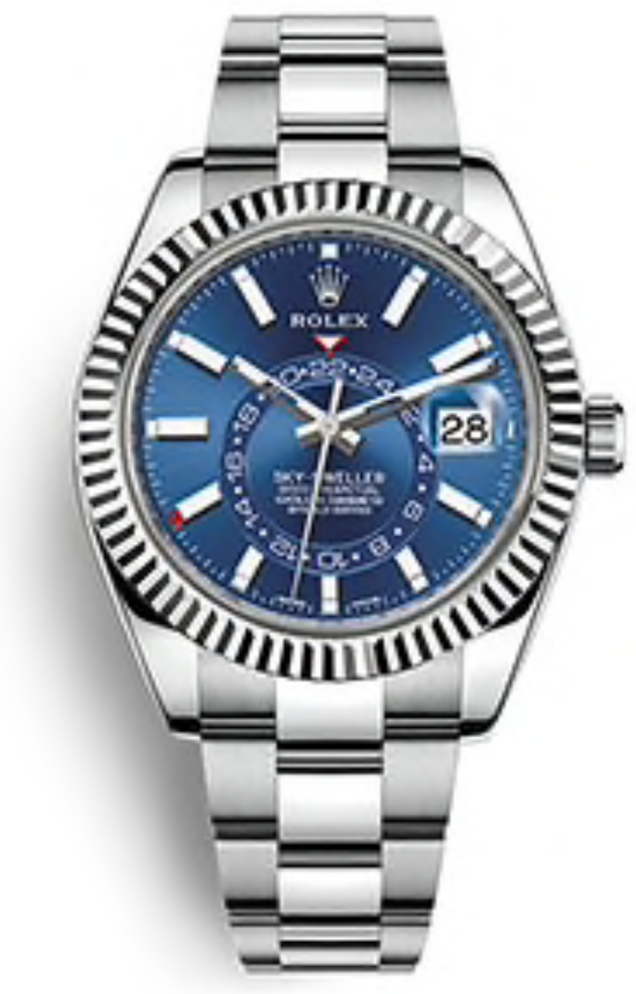
Point blocking [Jo and Kulkarni 2011, 2012; Weijiang et al. 2015] is the equivalent of tiling but for recursive tree programs. Tuning happens at runtime because “tile size” is input dependent

novel architectures

- **Hardware platforms often change dramatically**

novel architectures

- **Hardware platforms often change dramatically**

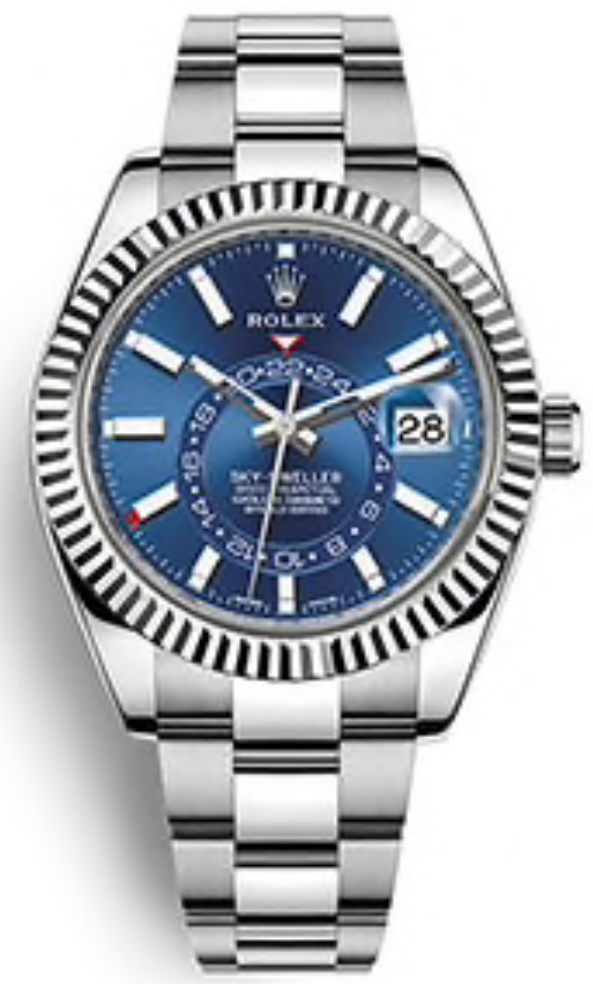


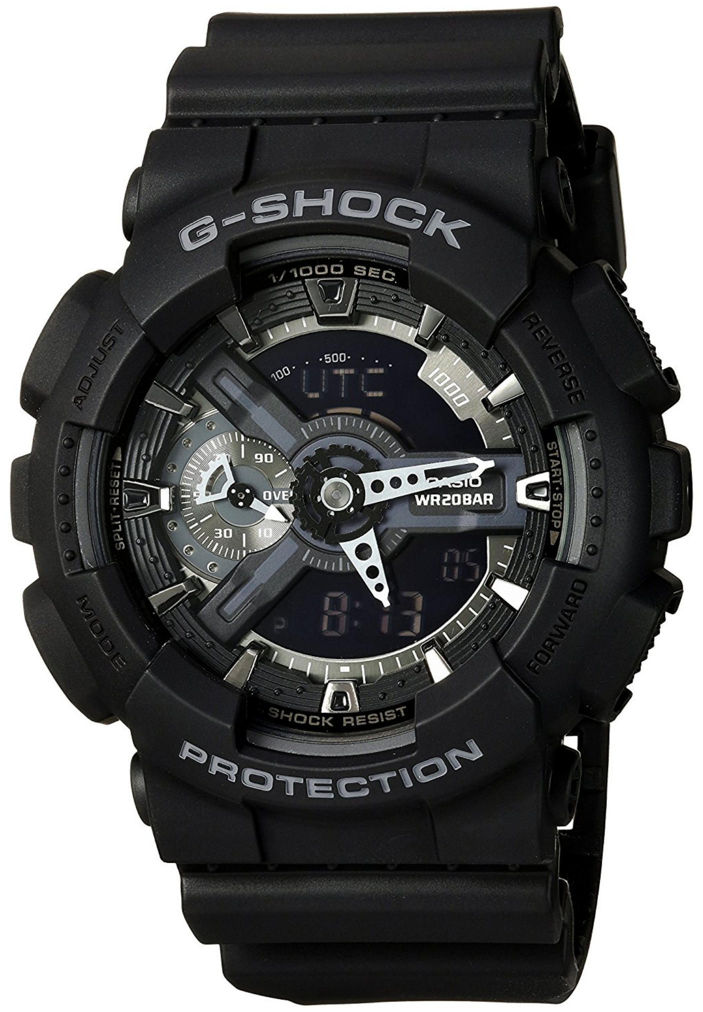
CRISP

PURDUE UNIVERSITY

novel architectures

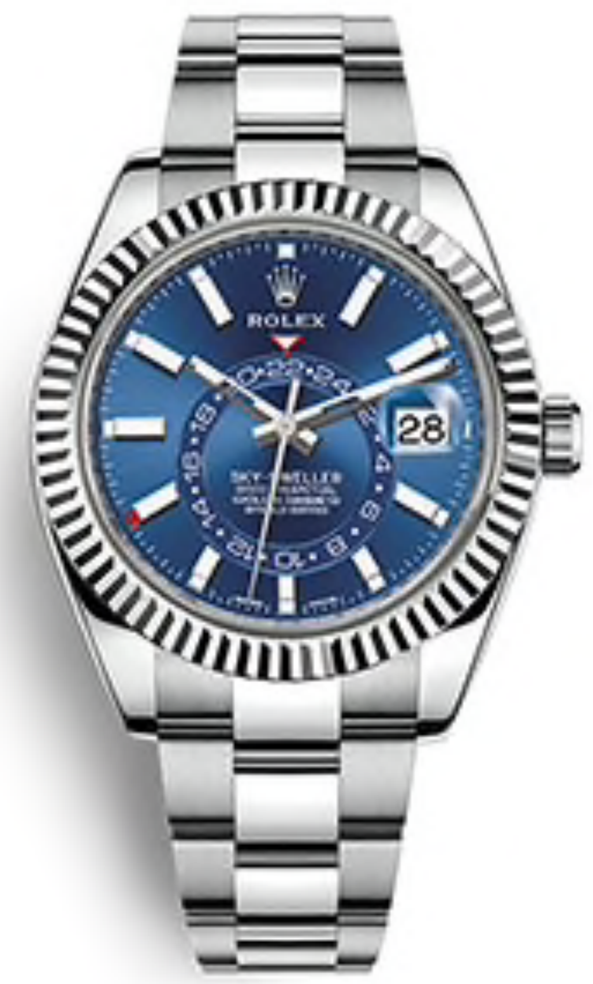
- **Hardware platforms often change dramatically**

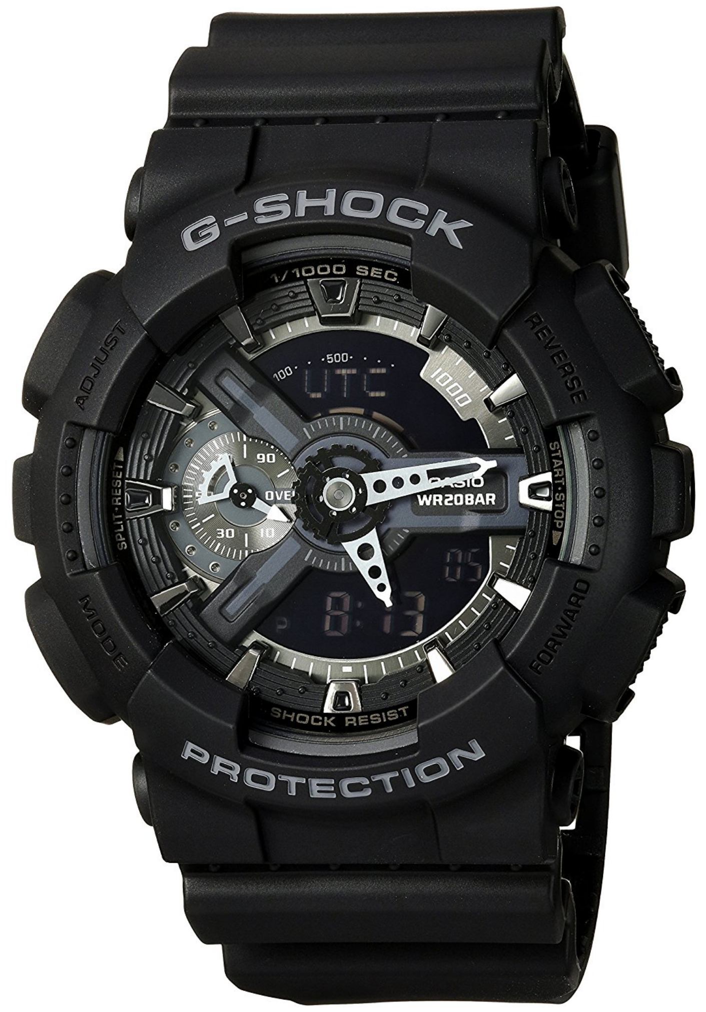


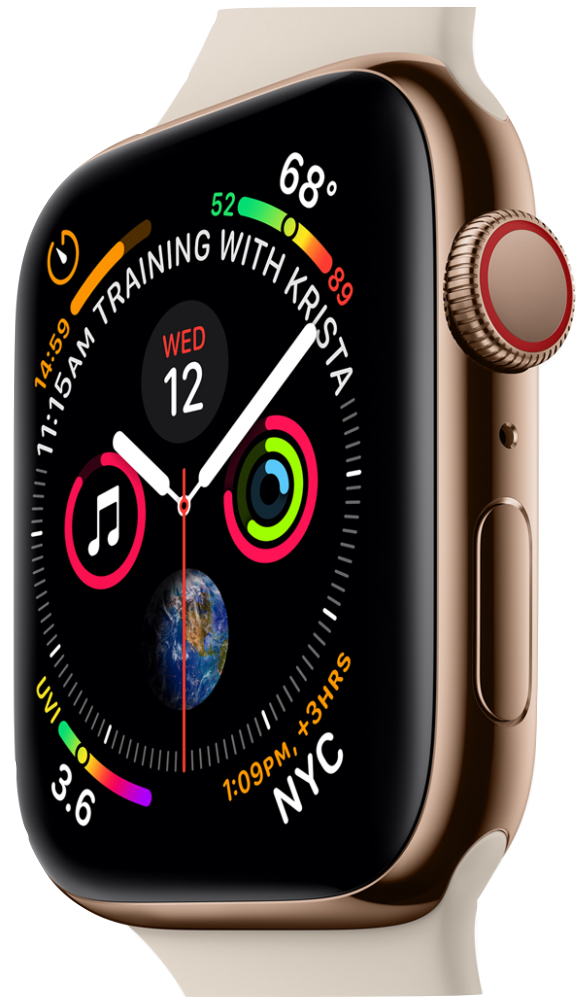


novel architectures

- **Hardware platforms often change dramatically**







novel architectures

- **Hardware platforms often change dramatically**
  - Software that is written for one set of assumptions that hold true for one type of hardware may not hold true for a different type of hardware
  - Modern GPUs allow programmers to easily write parallel programs
    - But simply porting over programs that worked well on CPUs does not work — GPUs have different execution models that must be managed to get good performance
  - Most work is compile-time or even earlier: compilers restructure code to work well on GPUs, or programmers use different design principles to write GPU code
  - **One challenge**: how can programmers make as few changes to their code as possible to have high performance GPUs

novel architectures

- **Hardware platforms often change dramatically**
  - Software that is written for one set of assumptions that hold true for one type of hardware may not hold true for a different type of hardware
  - Moder
    - But … not work — GPU … t good perf
  - Most w … to work well on GPUs, or programmers use different design principles to write GPU code
  - **One challenge**: how can programmers make as few changes to their code as possible to have high performance GPUs

Our work looks at algorithmic and compiler techniques to map *irregular* programs (that have unpredictable control and data behavior) to GPUs effectively [Jo and Kulkarni, 2013; Goldfarb and Kulkarni 2013; Ren et al 2015; Liu et al 2016; Ren et al 2017]

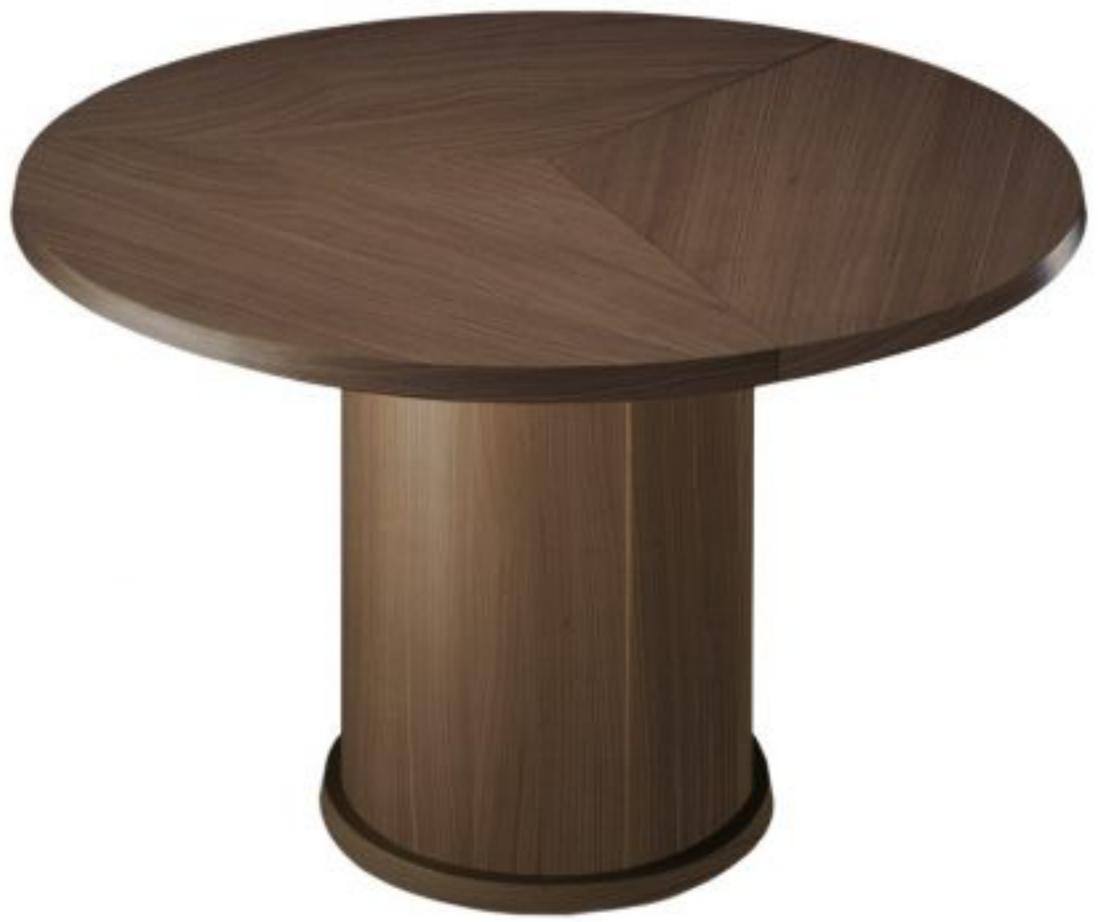
We have also looked at targeting distributed memory machines in the same way [Hegde et al 2017a, b]

elasticity

- Hardware resources change at runtime in a planned way

elasticity

- Hardware resources change at runtime in a planned way

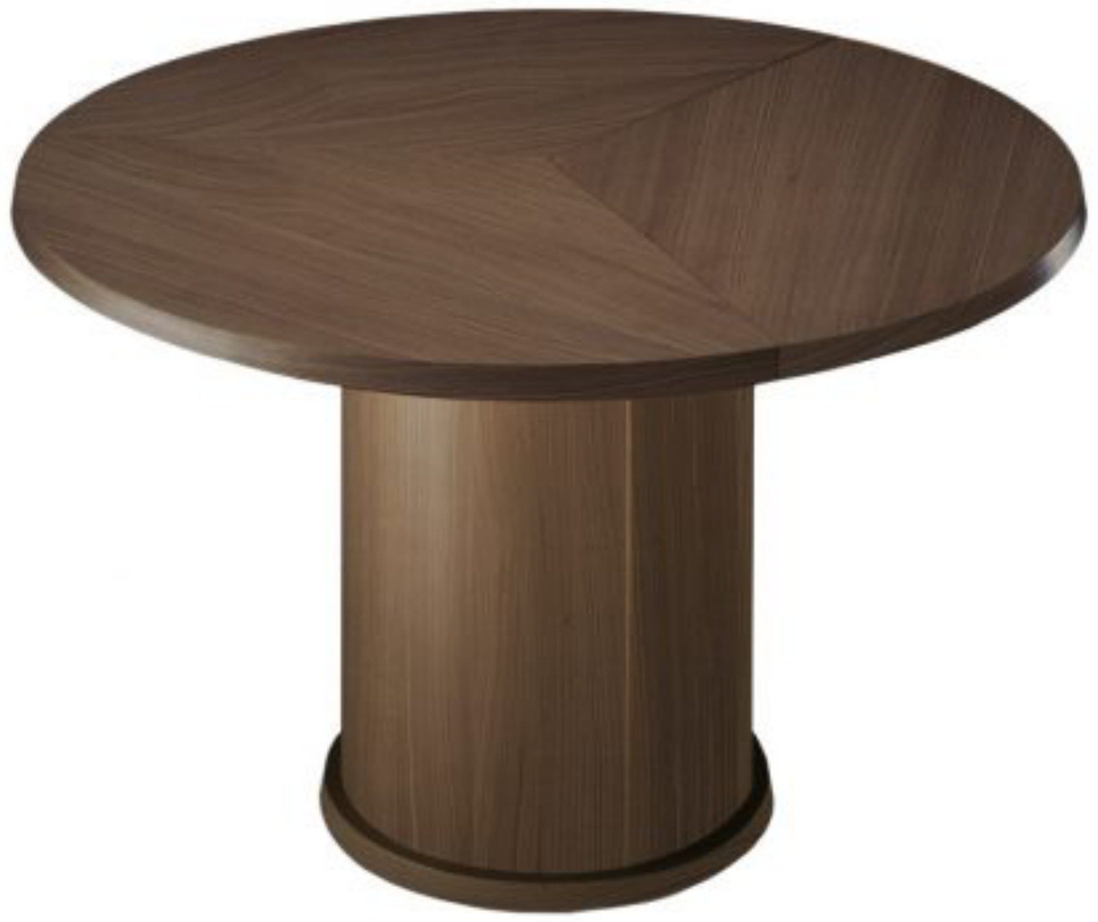


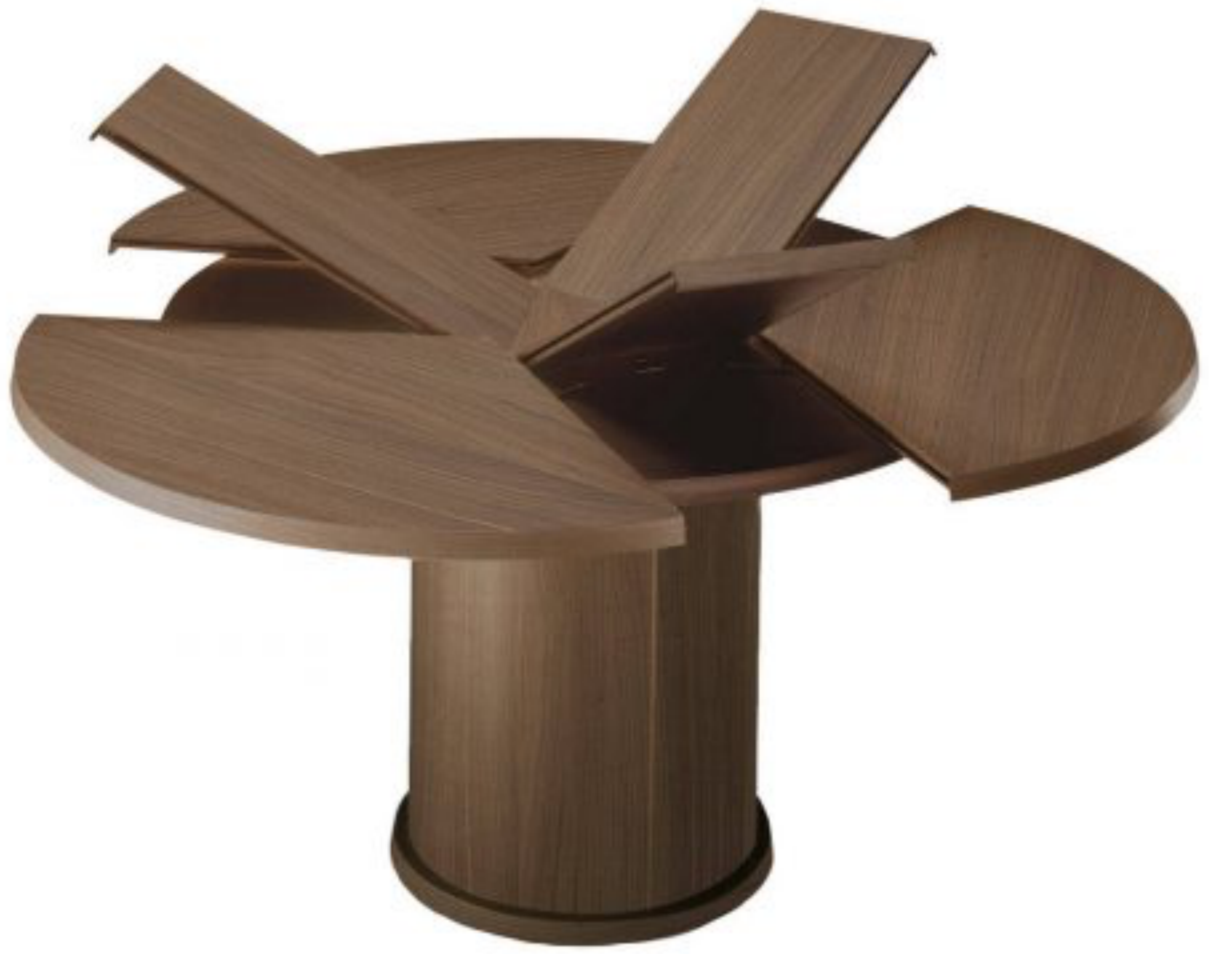
CRISP

PURDUE UNIVERSITY

elasticity

- Hardware resources change at runtime in a planned way



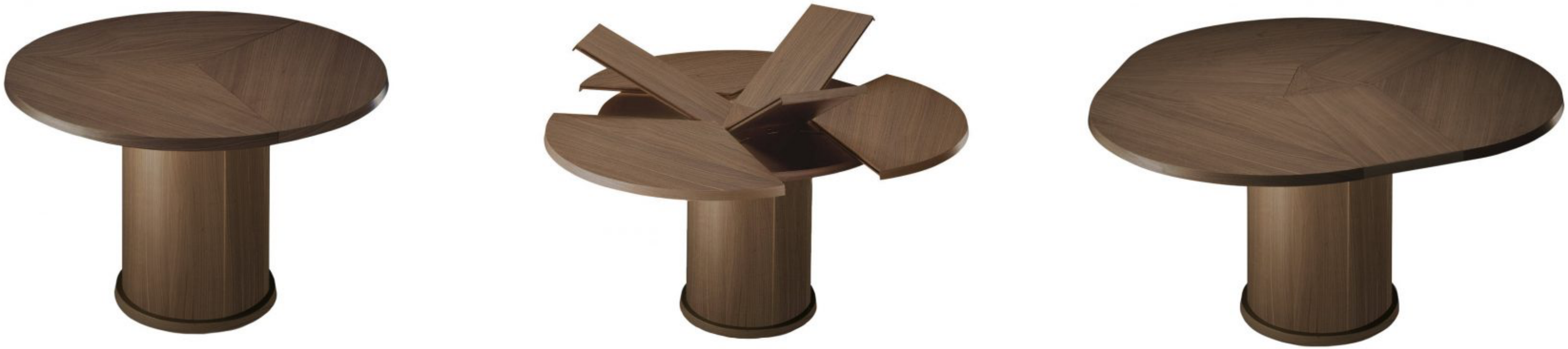


CRISP

PURDUE UNIVERSITY

elasticity

- **Hardware resources change at runtime in a planned way**



elasticity

- **Hardware resources change at runtime in a planned way**
  - Hardware resources get added or removed at runtime in response to, e.g., changing demand (think: cloud computing)
    - Key difference from fault tolerance: hardware change is planned
  - Software must be designed to dynamically adapt to different hardware resources
    - Should *take advantage* of those resources (performance should be proportional to resources)
    - Should happen without interruption/disruption of software behavior (i.e., without restarting software)
  - **One challenge**: how can we write software so programmers do not have to consider the challenges of implementing elasticity

elasticity

- **Hardware resources change at runtime in a planned way**
  - Hardware resources get added or removed at runtime in response to, e.g., changing demand (think: cloud computing)
    - Key
  - Softwa … e resources
    - Sho … prop
    - Sho … (i.e., without restarting software)
  - **One challenge**: how can we write software so programmers do not have to consider the challenges of implementing elasticity

InContext [Yoo et al. 2011] and EventWave [Chuang et al. 2013] provide programming models for *transparent elasticity*: programmers write *inelastic* software that does not consider dynamically changing hardware, and a runtime system transparently provides elasticity.

Key challenge: how to restrict the programming model as little as possible while providing transparent elasticity

input scaling

- **Unexpected (but normal) inputs may lead to unexpected outcomes**

input scaling

- Unexpected (but normal) inputs may lead to unexpected outcomes

input scaling

- Unexpected (but normal) inputs may lead to unexpected outcomes

input scaling

- **Unexpected (but normal) inputs may lead to unexpected outcomes**
  - Handling "out of the ordinary" inputs is part of the security problem, but what about inputs that a developer did not expect?
    - e.g., inputs much larger than expected?
  - If a developer has not tested against that kind of input, program may not do what they expect
    - Does a program still perform adequately? Does the program even run correctly at all?
  - **One challenge**: how do you know whether a program is behaving "incorrectly" when presented with an unexpected input?

input scaling

- **Unexpected (but normal) inputs may lead to unexpected outcomes**
  - Handling "out of the ordinary" inputs is part of the security problem, but what about inputs that a developer did not expect?
    - e.g.
  - If a de … m may not d…
    - Does a program still perform adequately? Does the program even run correctly at all?
  - **One challenge**: how do you know whether a program is behaving "incorrectly" when presented with an unexpected input?

Vrisha [Zhou et al, 2011], **Abhranta** [Zhou et al, 2013], and **WuKong** [Zhou et al, 2015] use machine learning to build models of program behavior as a function of input scale, and then use these models to determine if a program is behaving abnormally at large scales

many dimensions to adaptability

- **Design principle or (semi) automatic technique?**
- When should software adapt?
    - Compile time vs. start up time vs. continuous
- How do we introduce adaptability?
    - Manual vs. compiler-driven vs. run-time