More Data + Heterogeneous Compute = Fractured Workflows: Why mathematical and computational structure matter more than ever

Raymond J. Spiteri

Department of Computer Science
University of Saskatchewan

Go20 Conference on Scientific Computing and Software
18 May 2026

Data explosion

Data explosion

- **CERN : PB/day (hundreds of Gb/s sustained)**

Data explosion

- **CERN : PB/day (hundreds of Gb/s sustained)**
- **IoT : 10^{10} devices (always-on streams)**

Data explosion

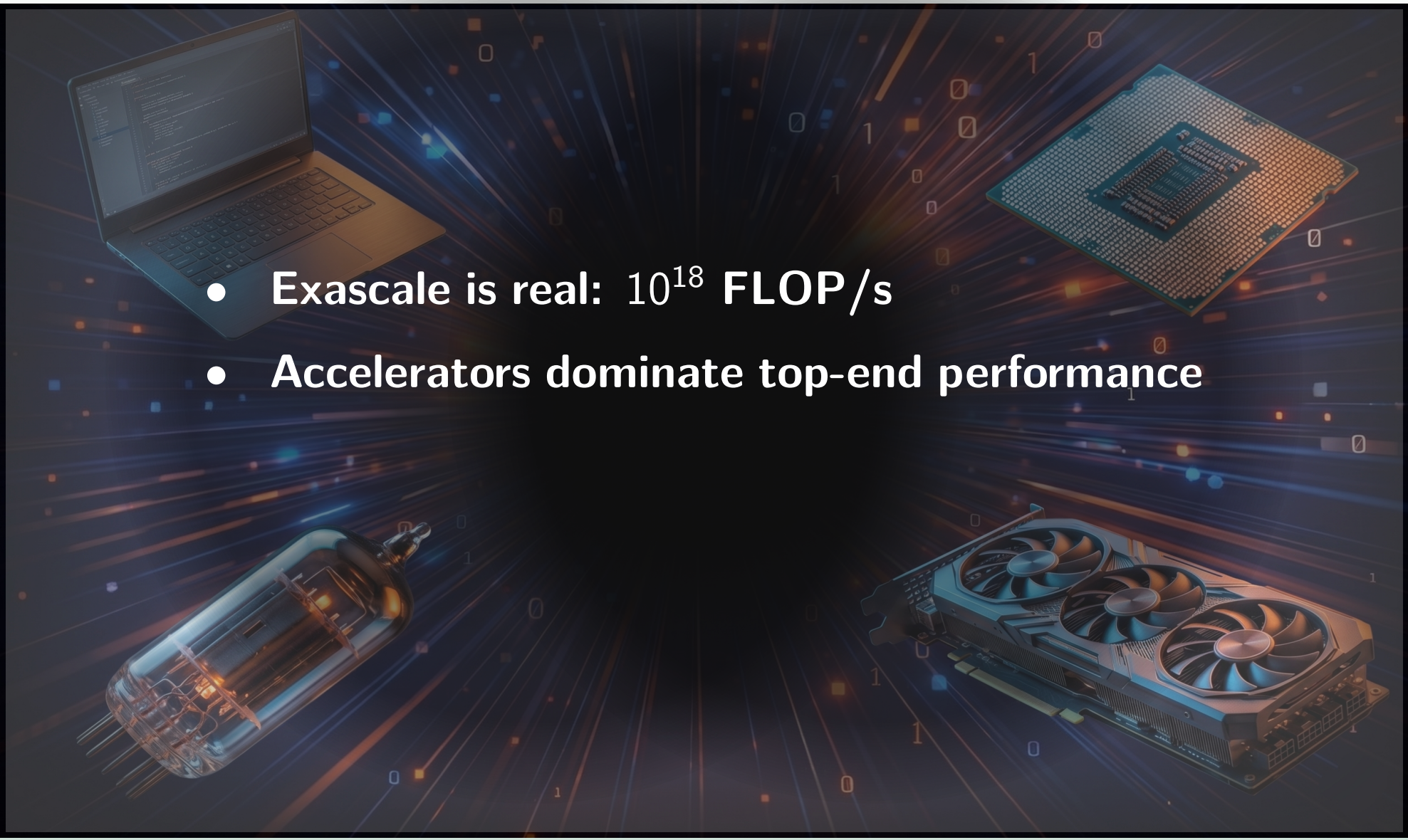
- **CERN : PB/day (hundreds of Gb/s sustained)**
- **IoT : 10^{10} devices (always-on streams)**
- **Global : ZB/year (EB/day scale)**

Compute explosion

Compute explosion

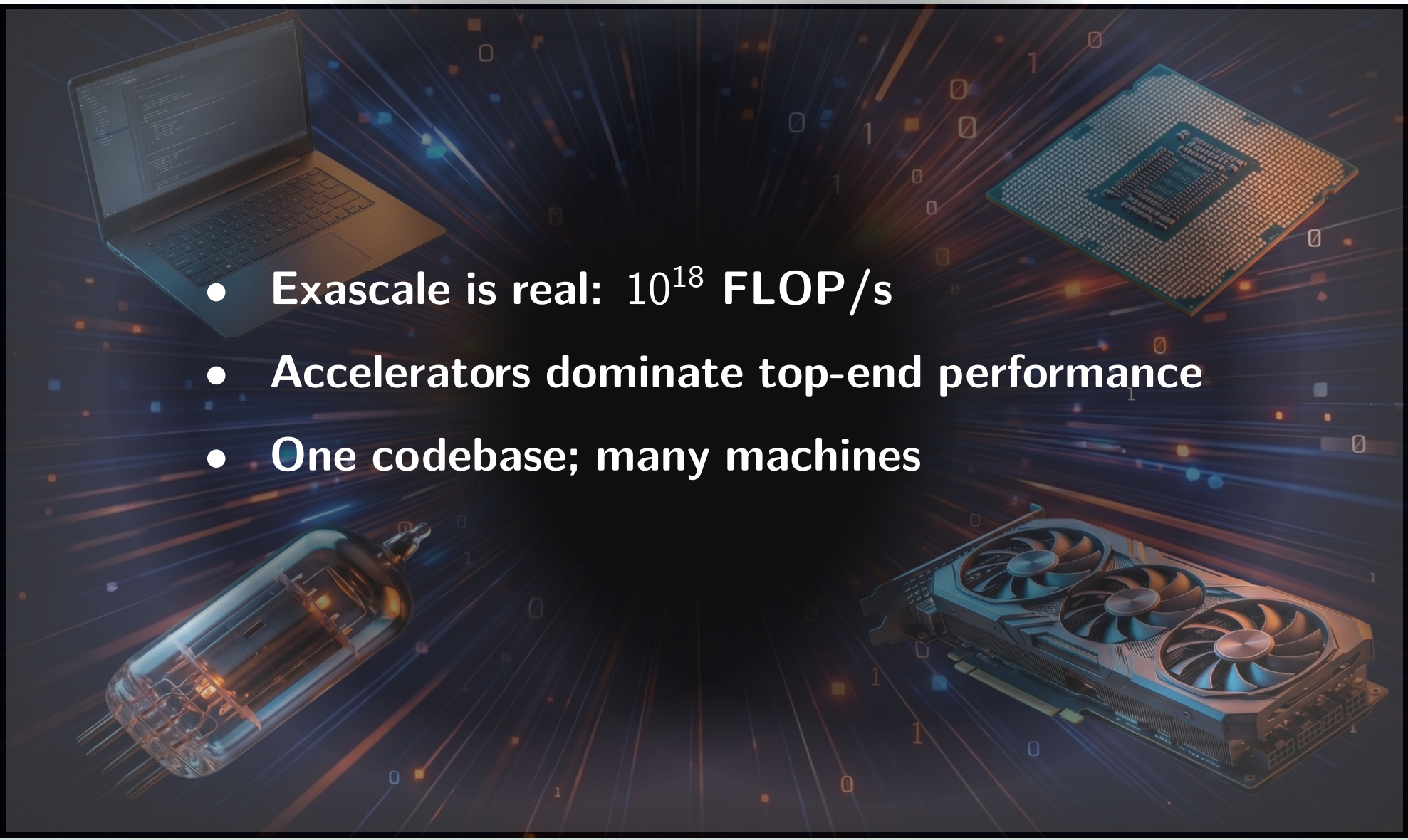
- **Exascale is real:** 10^{18} **FLOP/s**

Compute explosion



- **Exascale is real:** 10^{18} **FLOP/s**
- **Accelerators dominate top-end performance**

Compute explosion



- **Exascale is real: 10^{18} FLOP/s**
- **Accelerators dominate top-end performance**
- **One codebase; many machines**

A maxim

As machines become more powerful, the efficiency of algorithms grows more important, not less.

— Maxim 9, Trefethen, Maxims about numerical mathematics, science, computers, and life on earth

Another maxim

The only things that outpace the rate of growth in data and computing are . . .

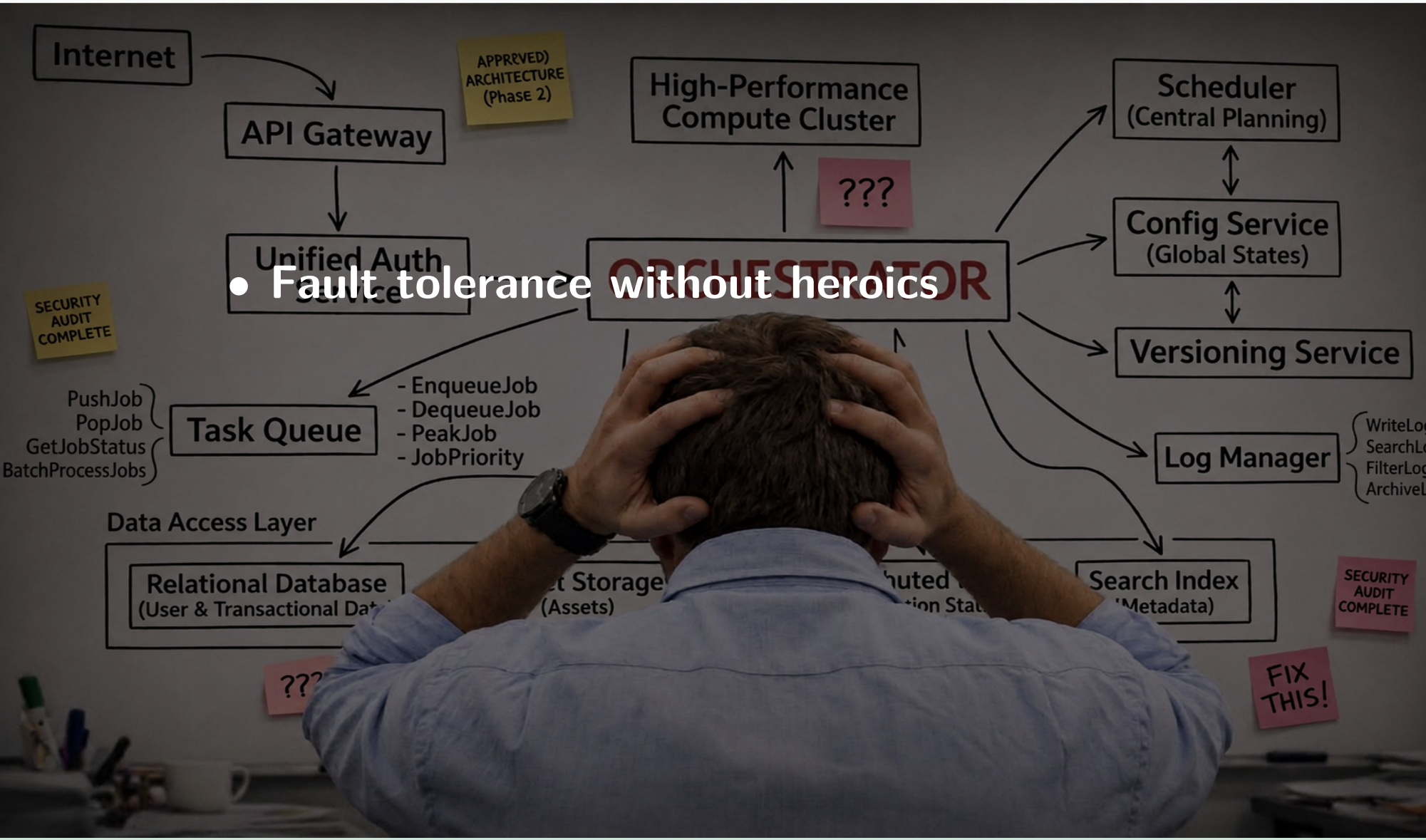
Another maxim

The only things that outpace the rate of growth in data and computing are . . .

our expectations for speed, fidelity, and reliability.

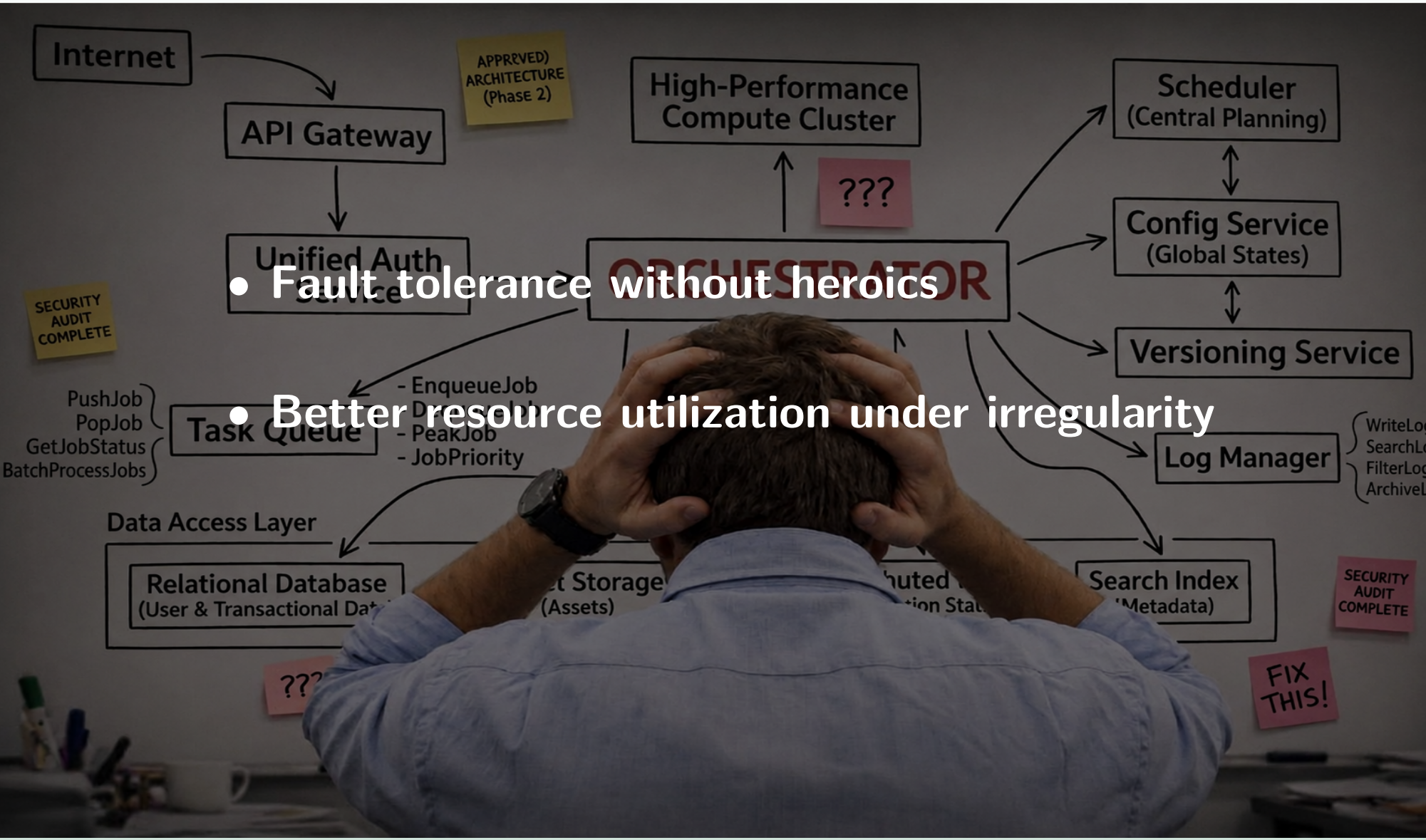
Workflows: what do we want

Workflows: what do we want



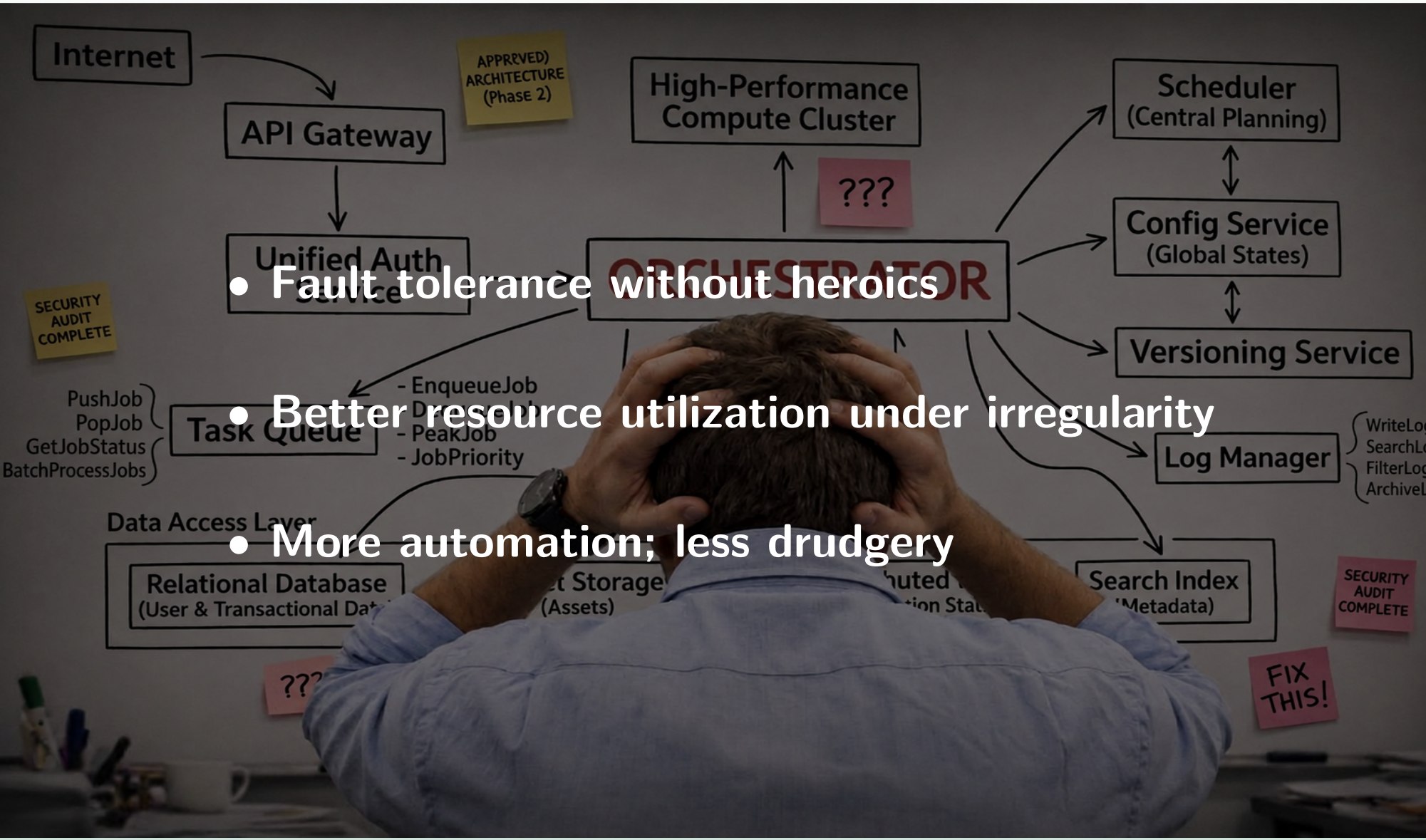
- **Fault tolerance without heroics**

Workflows: what do we want



- **Fault tolerance without heroics**
- **Better resource utilization under irregularity**

Workflows: what do we want



- **Fault tolerance without heroics**
- **Better resource utilization under irregularity**
- **More automation; less drudgery**

When things get messy

When things get messy, we simplify.

When things get messy

When things get messy, we simplify.

- split mathematically
- split computationally

When things get messy

When things get messy, we simplify.

- split mathematically
- split computationally

Useful instincts. Dangerous habits.

What splitting can break mathematically

We split because it simplifies.

What splitting can break mathematically

We split because it simplifies.

Fractional-step methods for differential equations

What splitting can break mathematically

We split because it simplifies.

Fractional-step methods for differential equations

- split processes
- advance them independently
- hope the important interactions survive

What are we really signing up for?

What splitting can break computationally

We split because it simplifies.

What splitting can break computationally

We split because it simplifies.

- split the work
- impose synchronization points
- wait, and hope the waiting is not too bad

What splitting can break computationally

We split because it simplifies.

- split the work
- impose synchronization points
- wait, and hope the waiting is not too bad

What are we really signing up for?

Not separate issues

Mathematical and computational structure shape one another.

Not separate issues

Mathematical and computational structure shape one another.

- how we split the mathematics shapes the computation
- how we orchestrate the computation shapes the mathematics
- accuracy, performance, and reliability are entangled

Not separate issues

Mathematical and computational structure shape one another.

- how we split the mathematics shapes the computation
- how we orchestrate the computation shapes the mathematics
- accuracy, performance, and reliability are entangled

Structure should holistically drive the method.

Example: Niederer benchmark

Monodomain model of cardiac electrophysiology

PDE:

$$\chi C_m \frac{\partial v}{\partial t} = \nabla \cdot \left(\frac{\lambda}{1+\lambda} \sigma_i \nabla v \right) - \chi \Big(I_{\text{ion}}(\mathbf{s}, v) + I_{\text{stim}}(t, \mathbf{x}) \Big)$$

$$\frac{\partial \mathbf{s}}{\partial t} = \mathbf{g}(\mathbf{s}, v)$$

Discretized and split:

$$\begin{bmatrix} \dot{\mathbf{V}} \\ \dot{\mathbf{S}} \end{bmatrix} = \begin{bmatrix} \frac{1}{C_m \chi} \sigma^{-1} \mathbf{D} \mathbf{V} \\ 0 \end{bmatrix} + \begin{bmatrix} -\frac{1}{C_m} (I_{\text{ion}}(\mathbf{S}, \mathbf{V}) + I_{\text{stim}}(t)) \\ \mathbf{G}(\mathbf{S}, \mathbf{V}) \end{bmatrix}$$

Example: Niederer benchmark

Monodomain model of cardiac electrophysiology

PDE:

$$\chi C_m \frac{\partial v}{\partial t} = \nabla \cdot \left(\frac{\lambda}{1+\lambda} \sigma_i \nabla v \right) - \chi \Big(I_{\text{ion}}(\mathbf{s}, v) + I_{\text{stim}}(t, \mathbf{x}) \Big)$$

$$\frac{\partial \mathbf{s}}{\partial t} = \mathbf{g}(\mathbf{s}, v)$$

Discretized and split:

$$\begin{bmatrix} \dot{\mathbf{V}} \\ \dot{\mathbf{S}} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{1}{C_m \chi} \sigma^{-1} \mathbf{D} \mathbf{V} \\ 0 \end{bmatrix}}_{\text{diffusion}} + \underbrace{\begin{bmatrix} -\frac{1}{C_m} (I_{\text{ion}}(\mathbf{S}, \mathbf{V}) + I_{\text{stim}}(t)) \\ \mathbf{G}(\mathbf{S}, \mathbf{V}) \end{bmatrix}}_{\text{reaction}}$$

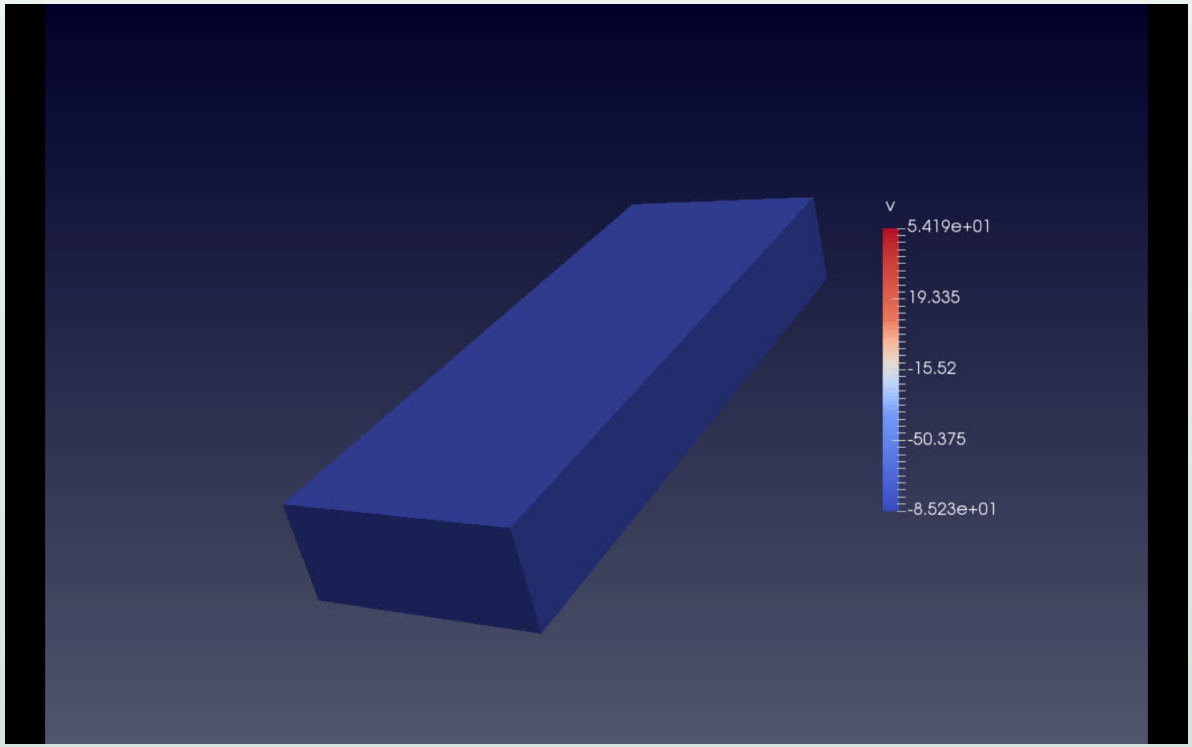
Example: Niederer benchmark details

Domain: $\mathbf{x} \in 0.3 \text{ cm} \times 0.7 \text{ cm} \times 2 \text{ cm}$; $t \in [0, 40]$ ms

Spatial discretization: central FD, $\Delta x = \Delta y = \Delta z = 0.05$ cm

Cell model: Ten Tusscher–Panfilov (19 ODEs; stiff)

Target $< 5\%$ $[\text{MRMS}]_v$ error



Example: Niederer benchmark

The split is natural. The design still matters.

Example: Niederer benchmark

The split is natural. The design still matters.

- same split
- different orchestration
  - different accuracies, stable step sizes
  - different run times

Example: Niederer benchmark

The split is natural. The design still matters.

- same split
- different orchestration
  - different accuracies, stable step sizes
  - different run times

Splitting is (by far) not the end of method design.

A framework for splitting method design

Theorem

An N-split FSRK method has an extended Butcher tableau

$$\begin{array}{c|c:c:c:c|c:c:c:c} \mathbf{c}^{[1]} & \mathbf{c}^{[2]} & \dots & \mathbf{c}^{[N]} & \mathbf{A}^{[1]} & \mathbf{A}^{[2]} & \dots & \mathbf{A}^{[N]} \\ \hline & & & & \mathbf{b}^{[1]} & \mathbf{b}^{[2]} & \dots & \mathbf{b}^{[N]} \end{array}$$

Theorem

The test equation $\frac{\mathrm{d}y}{\mathrm{d}t} = \sum_{\ell=1}^{N} \lambda^{[\ell]} y$ *leads to the stability function*

$$R(z^{[1]}, z^{[2]}, \dots, z^{[N]}) = \prod_{k=1}^{s} \prod_{\ell=1}^{N} R_k^{[\ell]}(\alpha_k^{[\ell]} z^{[\ell]}),$$

where $z^{[\ell]} = \Delta t \lambda^{[\ell]}$ *and* $R_k^{[\ell]}(z^{[\ell]})$ *is the stability function of the Runge–Kutta method at stage* k *applied to operator* ℓ.

A framework for splitting method design

Theorem

An N-split FSRK method has an extended Butcher tableau

$$\begin{array}{c|ccc|ccc} \mathbf{c}^{[1]} & \mathbf{c}^{[2]} & \cdots & \mathbf{c}^{[N]} & \mathbf{A}^{[1]} & \mathbf{A}^{[2]} & \cdots & \mathbf{A}^{[N]} \\ \hline & & & & \mathbf{b}^{[1]} & \mathbf{b}^{[2]} & \cdots & \mathbf{b}^{[N]} \end{array}$$

Theorem

The test equation $\frac{\mathrm{d}y}{\mathrm{d}t} = \sum_{\ell=1}^{N} \lambda^{[\ell]} y$ *leads to the stability function*

$$R(z^{[1]}, z^{[2]}, \ldots, z^{[N]}) = \prod_{k=1}^{s} \prod_{\ell=1}^{N} R_k^{[\ell]}(\alpha_k^{[\ell]} z^{[\ell]}),$$

where $z^{[\ell]} = \Delta t \lambda^{[\ell]}$ *and* $R_k^{[\ell]}(z^{[\ell]})$ *is the stability function of the Runge–Kutta method at stage* k *applied to operator* ℓ.

A framework for splitting method design

Consequences:

A framework for splitting method design

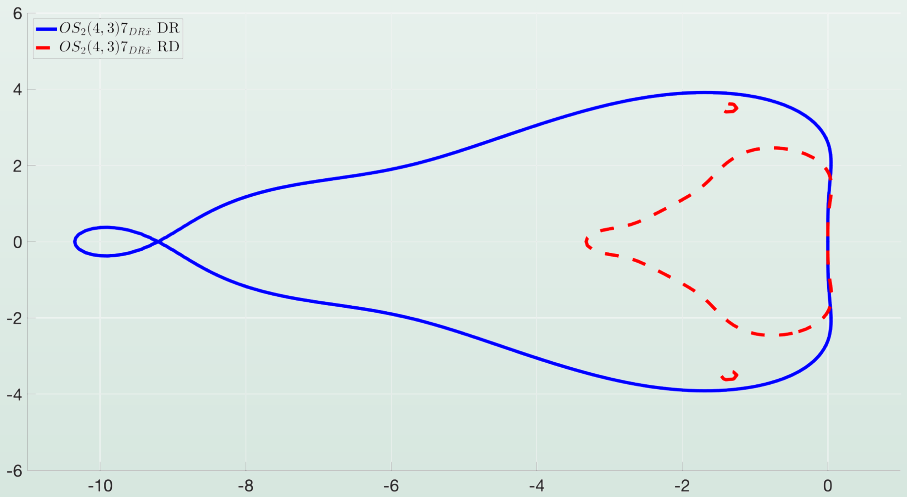
Consequences:

- sub-integrator choice matters
- operator ordering matters
- stability design enables larger stable steps

A framework for splitting method design

For the Niederer benchmark,

$$\lambda^{[D]} \approx -1.92,\ \lambda^{[R]} \approx -1260 \quad \Longrightarrow \quad |\lambda^{[D]}| \approx 0.001\,|\lambda^{[R]}|$$



Niederer benchmark: performance

Same split. Different design. Different outcome.

method	DR ordering			RD ordering		
	Δt	CPU (s)	$[\text{MRMS}]_v$	Δt	CPU (s)	$[\text{MRMS}]_v$
Ruth	2.8×10^{-3}	8831	6.7×10^{-4}	6.2×10^{-3}	3962	3.9×10^{-4}
AKS3	3.1×10^{-3}	7654	2.3×10^{-2}	3.1×10^{-3}	7969	2.2×10^{-2}
$\text{OS}_2(4,3)\,[7]_{DR\hat{x}}$	1.1×10^{-2}	2509	5.5×10^{-4}	—	—	—

Niederer benchmark: performance

Same split. Different design. Different outcome.

method	DR ordering			RD ordering		
	Δt	CPU (s)	$[\mathrm{MRMS}]_v$	Δt	CPU (s)	$[\mathrm{MRMS}]_v$
Ruth	2.8×10^{-3}	8831	6.7×10^{-4}	6.2×10^{-3}	3962	3.9×10^{-4}
AKS3	3.1×10^{-3}	7654	2.3×10^{-2}	3.1×10^{-3}	7969	2.2×10^{-2}
$\mathrm{OS}_2(4,3)\,[7]_{DR\hat{x}}$	1.1×10^{-2}	2509	5.5×10^{-4}	—	—	—

Theory-guided design turns into speed.

Example 2: Strength-Interval Curves

- measures recovery of tissue excitability
- useful for studying vulnerability to re-entry
- can be measured using S1–S2 protocol
- plot stimulus threshold for second wave vs. S2 interval
- one SI curve = many threshold solves

Example 2: Strength-Interval Curves

This is not a one-off simulation.

Example 2: Strength-Interval Curves

This is not a one-off simulation.

- many threshold solves
- lots of opportunity to waste time/effort/compute
  - similar work across intervals
- lots of opportunity to orchestrate
  - shared information across intervals

Example 2: Strength-Interval Curves

This is not a one-off simulation.

- many threshold solves
- lots of opportunity to waste time/effort/compute
  - similar work across intervals
- lots of opportunity to orchestrate
  - shared information across intervals

The workflow is the bottleneck.

What is an SI curve, really?

One SI curve is a family of related event-location problems.

What is an SI curve, really?

One SI curve is a family of related event-location problems.

What is an SI curve, really?

One SI curve is a family of related event-location problems.

- find minimum S1 to establish initial wave
- for each interval, find minimum S2 that re-triggers wave
- each threshold found by bisection

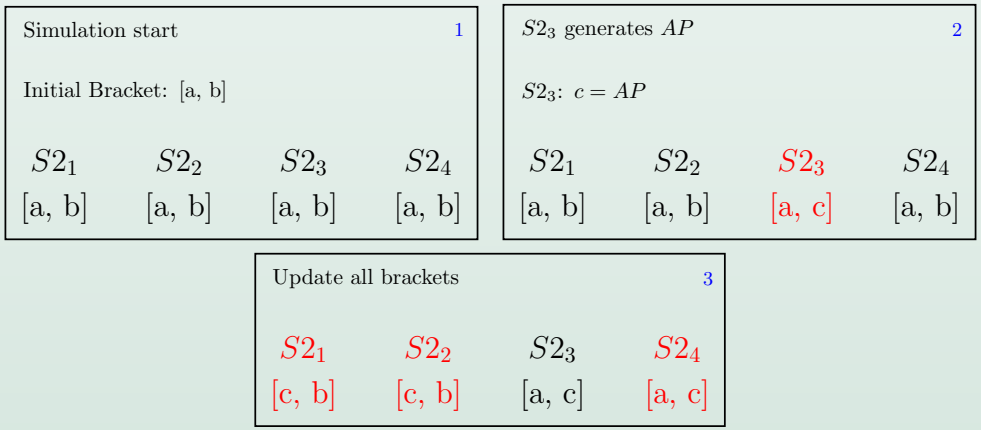
Why the workflow dominates

Lots of opportunity to orchestrate.

Why the workflow dominates

Lots of opportunity to orchestrate.

- compute S2 for each interval in parallel
- pick off initial condition from S1 simulation
- terminate once activation is detected
- communicate bracket information across intervals

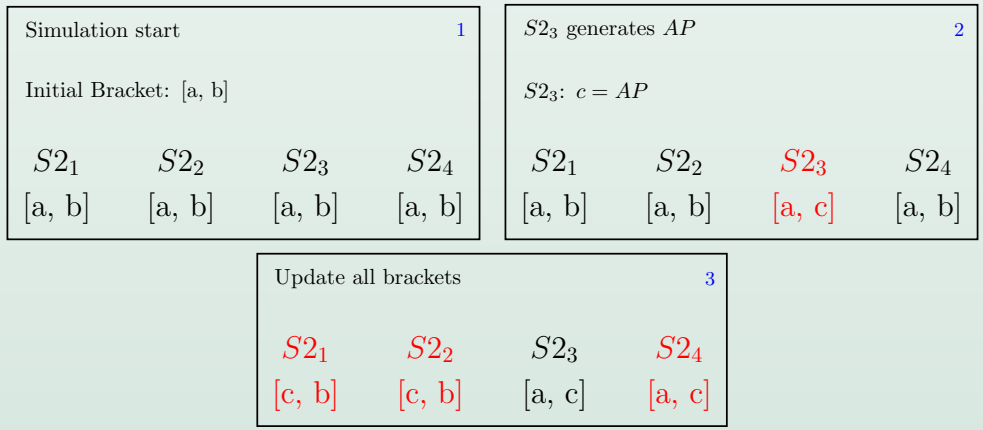


- use spare compute for lookahead bisection

Why the workflow dominates

Lots of opportunity to orchestrate.

- compute S2 for each interval in parallel
- pick off initial condition from S1 simulation
- terminate once activation is detected
- communicate bracket information across intervals



- use spare compute for lookahead bisection

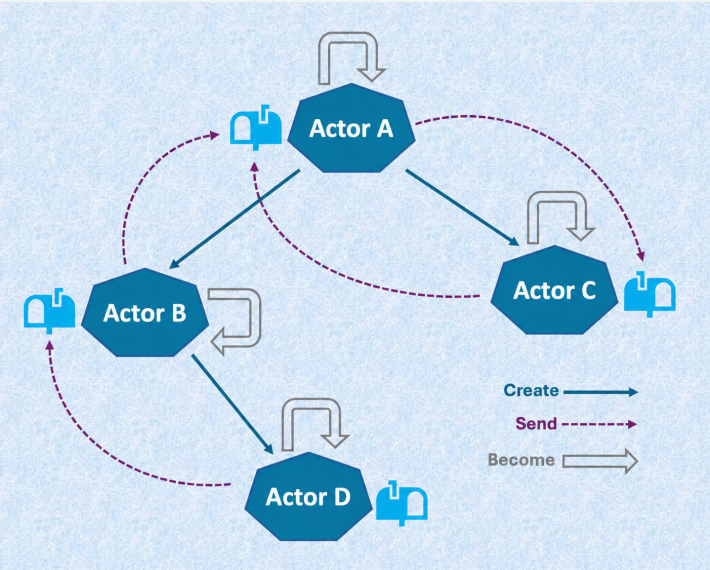
A painful fit for traditional computing.

Actors: a different orchestration model

The actor model of concurrent computation.

Actors: a different orchestration model

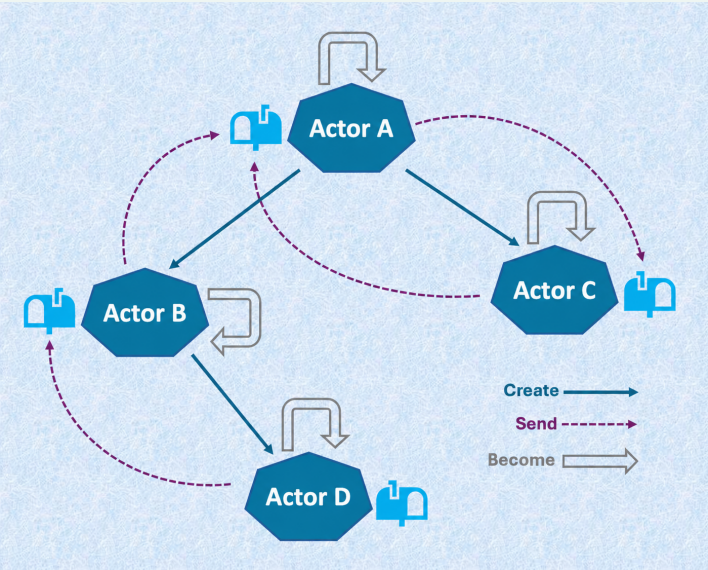
The actor model of concurrent computation.



- each actor
  - has **private state**
  - communicates by **messages**
  - executes **asynchronously**
- primitive operations:
  create, **send**, **become**

Actors: a different orchestration model

The actor model of concurrent computation.



- each actor
  - has **private state**
  - communicates by **messages**
  - executes **asynchronously**
- primitive operations:
  create, **send**, **become**

Coordination by messages, not by global lockstep.

Why do actors fit?

The computation is naturally asynchronous.

Why do actors fit?

The computation is naturally asynchronous.

- each interval progresses at its own pace
- updated brackets communicated once discovered
- irrelevant work killed
- spare resources redirected immediately

Show me the money

Time-to-solution for one SI curve:

Show me the money

Time-to-solution for one SI curve:

4,068 hours $\rightarrow$ 1 thread + serial bisection

Show me the money

Time-to-solution for one SI curve:

4,068 hours $\rightarrow$ 1 thread + serial bisection

50.3 hours $\rightarrow$ parallel OpenCARP

Show me the money

Time-to-solution for one SI curve:

4,068 hours $\rightarrow$ 1 thread + serial bisection

50.3 hours $\rightarrow$ parallel OpenCARP

26.8 hours $\rightarrow$ basic workflow optimizations

Show me the money

Time-to-solution for one SI curve:

4,068 hours	$\rightarrow$	1 thread + serial bisection
50.3 hours	$\rightarrow$	parallel OpenCARP
26.8 hours	$\rightarrow$	basic workflow optimizations
14.7 hours	$\rightarrow$	early stop, bracket update, lookahead

Show me the money

Time-to-solution for one SI curve:

4,068 hours	$\rightarrow$	1 thread + serial bisection
50.3 hours	$\rightarrow$	parallel OpenCARP
26.8 hours	$\rightarrow$	basic workflow optimizations
14.7 hours	$\rightarrow$	early stop, bracket update, lookahead

Better orchestration cuts days to hours.

Show me the money

Time-to-solution for one SI curve:

4,068 hours	$\rightarrow$	1 thread + serial bisection
50.3 hours	$\rightarrow$	parallel OpenCARP
26.8 hours	$\rightarrow$	basic workflow optimizations
14.7 hours	$\rightarrow$	early stop, bracket update, lookahead

Better orchestration cuts days to hours.

More utilization. Less drudgery.

What structure buys us

Reduce time to solution by respecting structure.

What structure buys us

Reduce time to solution by respecting structure.

- mathematical structure improves stability and accuracy
- computational structure improves utilization and robustness
- together, they reduce manual orchestration

What structure buys us

Reduce time to solution by respecting structure.

- mathematical structure improves stability and accuracy
- computational structure improves utilization and robustness
- together, they reduce manual orchestration

Move the human out of the critical path.

A parting wish

If you had a magic wand. . .

A parting wish

If you had a magic wand. . .

. . . what one scientific-computing problem would you wish away?

A parting wish

If you had a magic wand. . .

. . . what one scientific-computing problem would you wish away?

All the babysitting.

A parting wish

If you had a magic wand. . .

. . . what one scientific-computing problem would you wish away?

Less babysitting. More discovery.