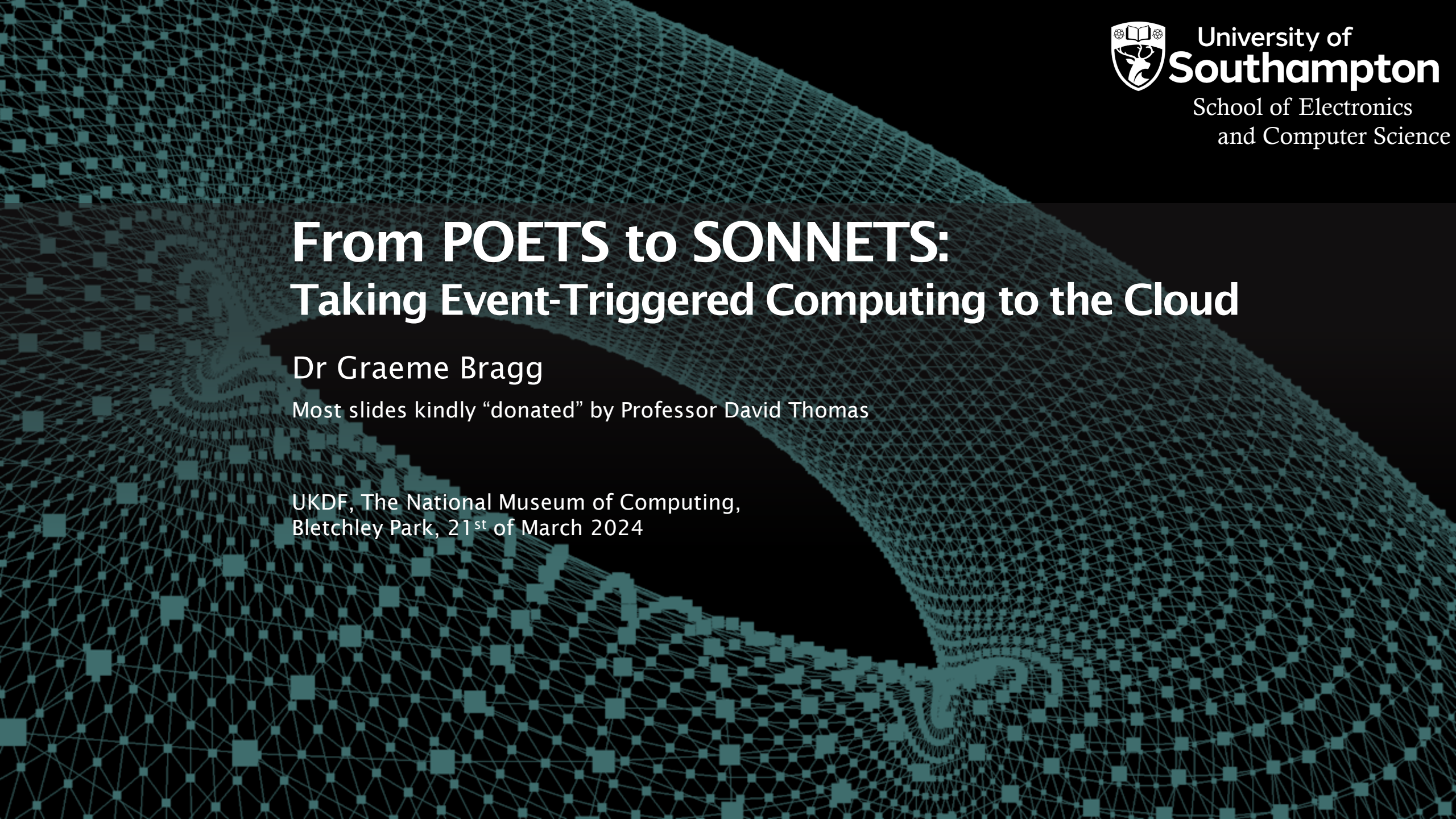


University of Southampton
School of Electronics and Computer Science

From POETS to SONNETS:
Taking Event-Triggered Computing to the Cloud

Dr Graeme Bragg

Most slides kindly “donated” by Professor David Thomas

UKDF, The National Museum of Computing,
Bletchley Park, 21st of March 2024

POETS : What ~~is~~ was it?

- EPSRC £5M Programme Grant
  - Running 2016 till 2022
- Four university partners:
  - Cambridge
  - Imperial
  - Newcastle
  - Southampton

POETS : The big idea

"Create a new framework for developing and executing event-triggered applications using asynchronous algorithms in distributed hardware"

Event-triggered = millions of shared-nothing threads sending tiny messages

- Research challenges:
  - ***Applications***: what should event-first algorithms look like?
  - ***Languages:*** what language do we use?
  - ***Compilation***: how do we describe and compile such applications?
  - ***Hardware***: what does this distributed hardware look like?

POETS : What we achieved over six years

- **Applications**: portfolio of asynchronous event-based applications
  - Flagship is “Dissipative Particle Dynamics (DPD)”
  - Allowed us to provide large speed-up for published computational chemistry research

- **Compilers**: multiple compilers and simulators for one language
  - Performs place and route for applications with 1M+ logical threads
  - Includes run-time management for input and output

- **Architecture**: bespoke CPU architecture and network called “Tinsel”
  - Custom RISC-V architecture with deeply embedded routed network
  - Supports 50K hardware threads in a low-latency communication mesh

ETC: Event Triggered Computing

- Applications are split into ***devices***
  - "device" = finite state machine
- Device state is a tiny part of the global state
- Devices interact through ***event*** handlers:
  - ***Receive***: a message *m* is sent to device *d*

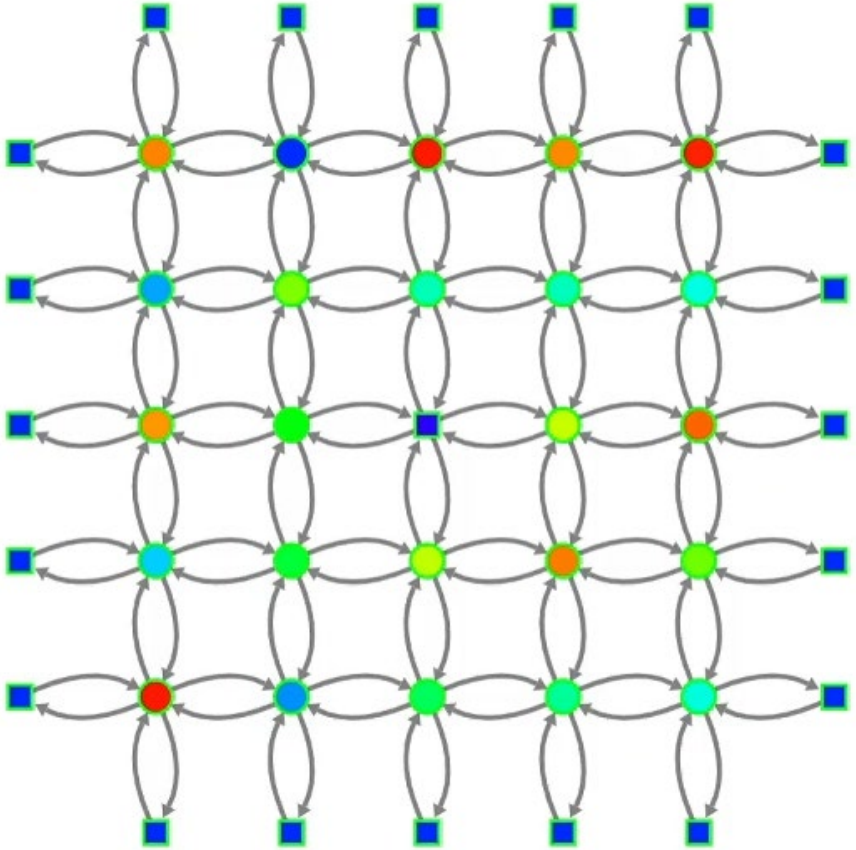
    d' = receive_handler(*d*, *m*)
  - ***Send***: device *d* sends a message *m*

    (*d'*,*m*) = send_handler(*d*)

State changes only occur on send or receive

All state changes are atomic

Hardware schedules both sends and receives



ETC: Inversion of control

```
class MyDevice
{
  int state;

  void run()
  {
        while(1){
            msg = recv();
            state = receive_handler(state, msg);

            while( more_messages(state) ){
                (state,msg) = send_handler(state);
                send( msg );
            }
        }
    }
};
```

Software is “in control”
Devices are finite state
Hardware buffering is un-bounded

```
class MyDevice
{
    int state;

    void on_recv(const Message &msg)
    {
        state = receive_handler(state, msg);
    }

    bool ready_to_send() const
    {
        return more_messages(state);
    }

    void on_send(Message &msg)
    {
        (state,msg) = send_handler(state);
    }
};
```

Hardware+compiler is “in control”
Software must wait for network capacity
Both devices and hardware can be finite state

Simple Graph Example: Single Source Shortest Path

- Messages give distance from source over edge
- State is the current distance to source
- Receive message:
  - Calculate transitive distance from target
  - Update vertex to new distance if it is lower
- Ready to send if new distance not yet shared
- Send message:
  - Send message to share new distance
  - Mark state as not ready to send

```
struct Message {
  int distance;
};

struct State {
  int currDistance;
  bool isDirty;

  void on_recv(const Message &msg) {
    int newDistance = msg.distance + 1;
    if (newDistance < currDistance) {
      currDistance=newDistance;
      isDirty=true;
    }
  }

  bool ready_to_send()
  { return isDirty; }

  void on_send(Message &msg) {
    msg.distance = currDistance;
    isDirty = false;
  }
};
```

Application model : semantics

- The ETC execution model is primarily mathematical
  - Described in terms of sets and functions
  - Formalised in TLA+ and Coq
- Sometimes quite loose: ***favour hardware efficiency***
  - Message sending is opportunistic: hardware decides when software is allowed to send
  - No guarantees on message delivery order: arbitrary re-ordering is allowed
- Sometimes quite strong: ***enable software***
  - Atomic broadcasts: any message sent eventually delivered to all recipients
  - All messages will *eventually* be delivered: no message loss

University of Southampton

Challenges

- Asynchronous software design is… hard
  - Message ordering is non-deterministic
  - Can't rely on a shared global clock

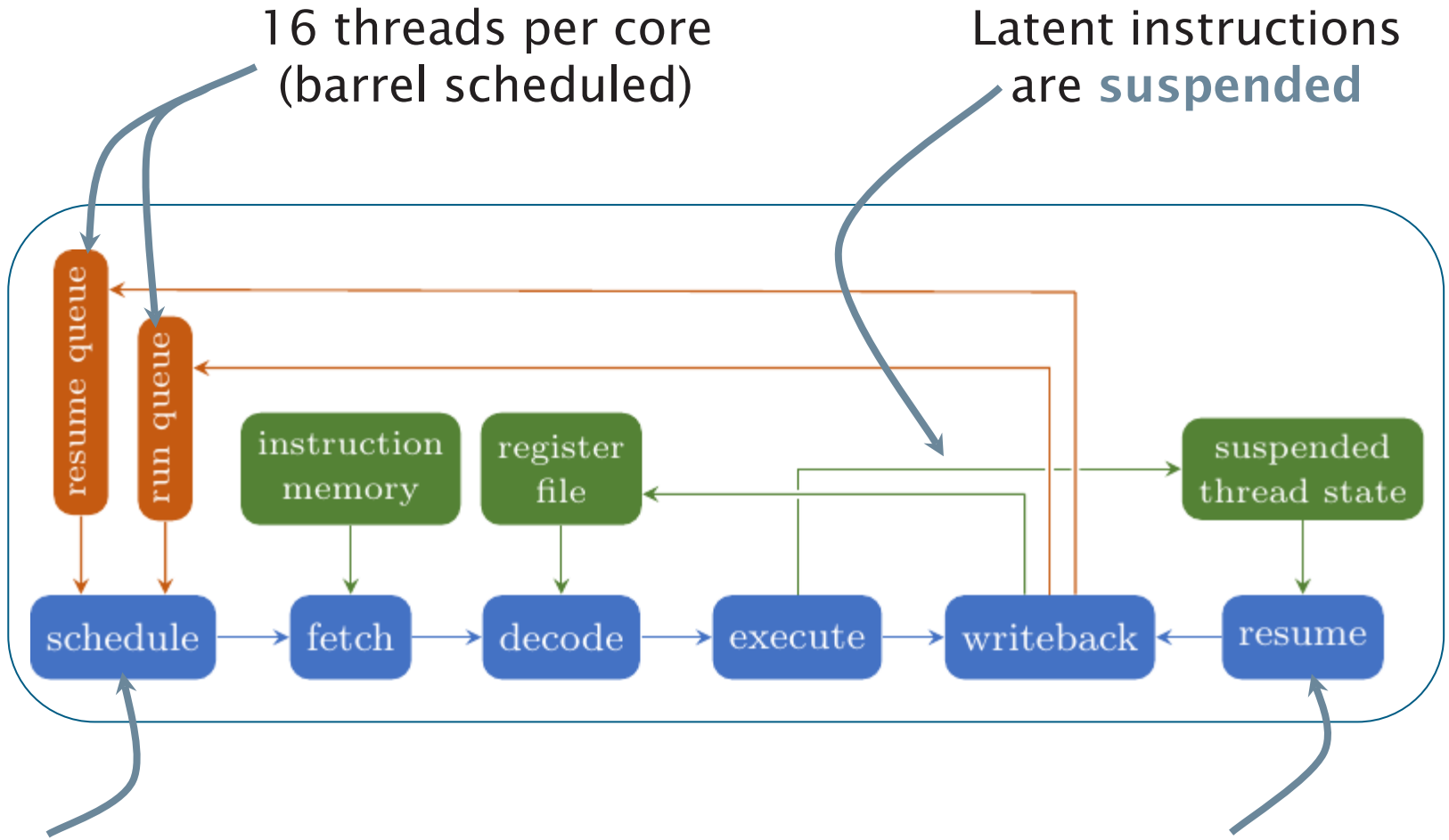
- We have to solve a number of big problems
  - ***Performance***: creating enough parallelism
  - ***Correctness***: avoiding deadlock and livelock
  - ***Exfiltration***: getting the answer out

University of Southampton

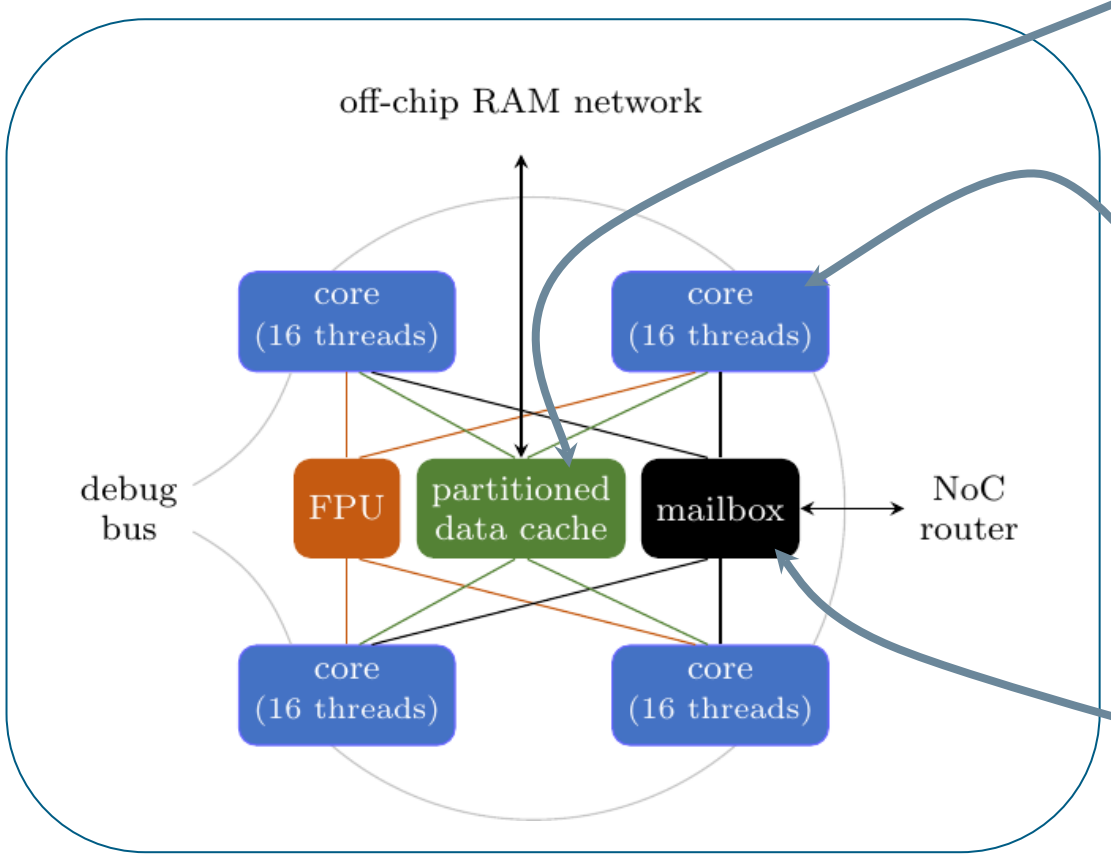
What should ETC hardware look like?

- Opportunities : the ETC model is designed to enable scalable hardware
  - Millions of asynchronous devices (small FSMs) provides scheduling freedom
  - Loose messaging and execution timing allows for latency hiding
  - Very constrained ETC software model reduces need for full OS services
  - Small messages : typically 16-64 bytes
- Constraints : the ETC model pushes all messaging and scheduling into hardware
  - Thousands of small FSMs need to be scheduled on each “CPU”
  - Messaging needs to be integrated at a deep level
  - Compute schedule is driven by unpredictable message arrival times
  - *Practically: it had to be something we could prototype and evaluate using lots of FPGAs*

Tinsel core: multithreaded RV32IMF



Tinsel tile: FPUs, caches, mailboxes



Data cache: **no global shared memory**

Custom instructions for **message-passing**

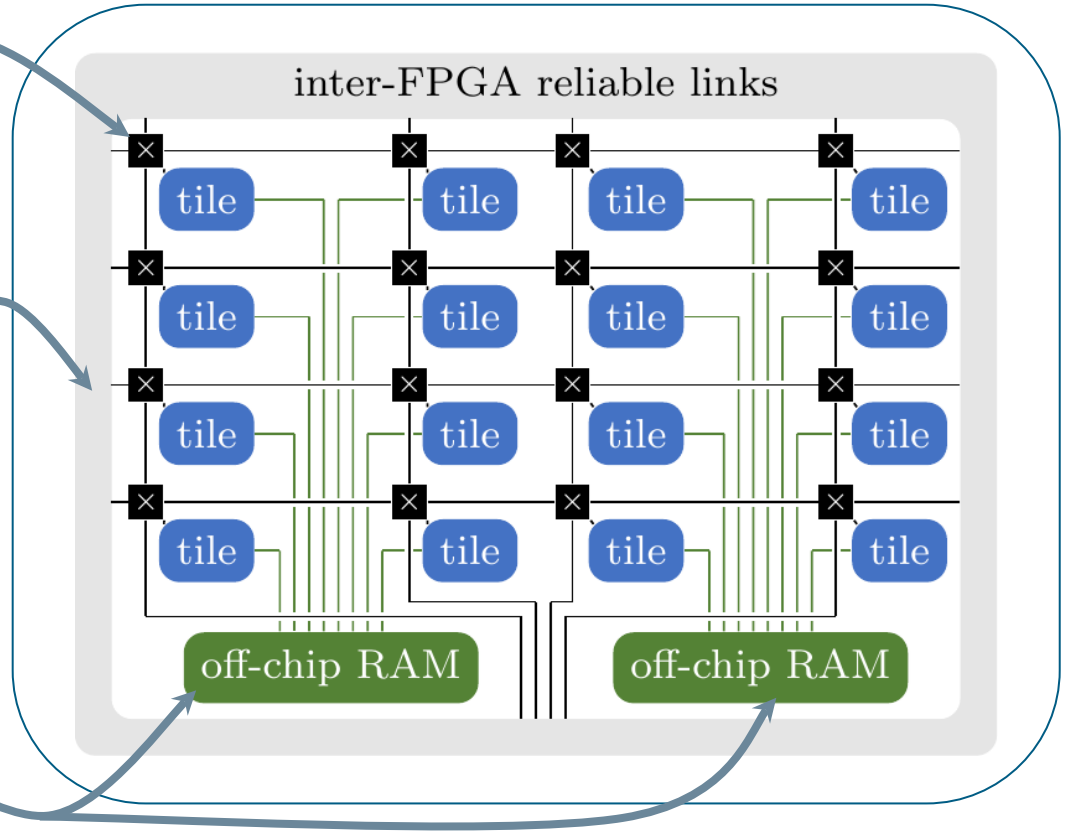
Mixed-width memory-mapped scratchpad

Tinsel network-on-chip

2D dimension-ordered router

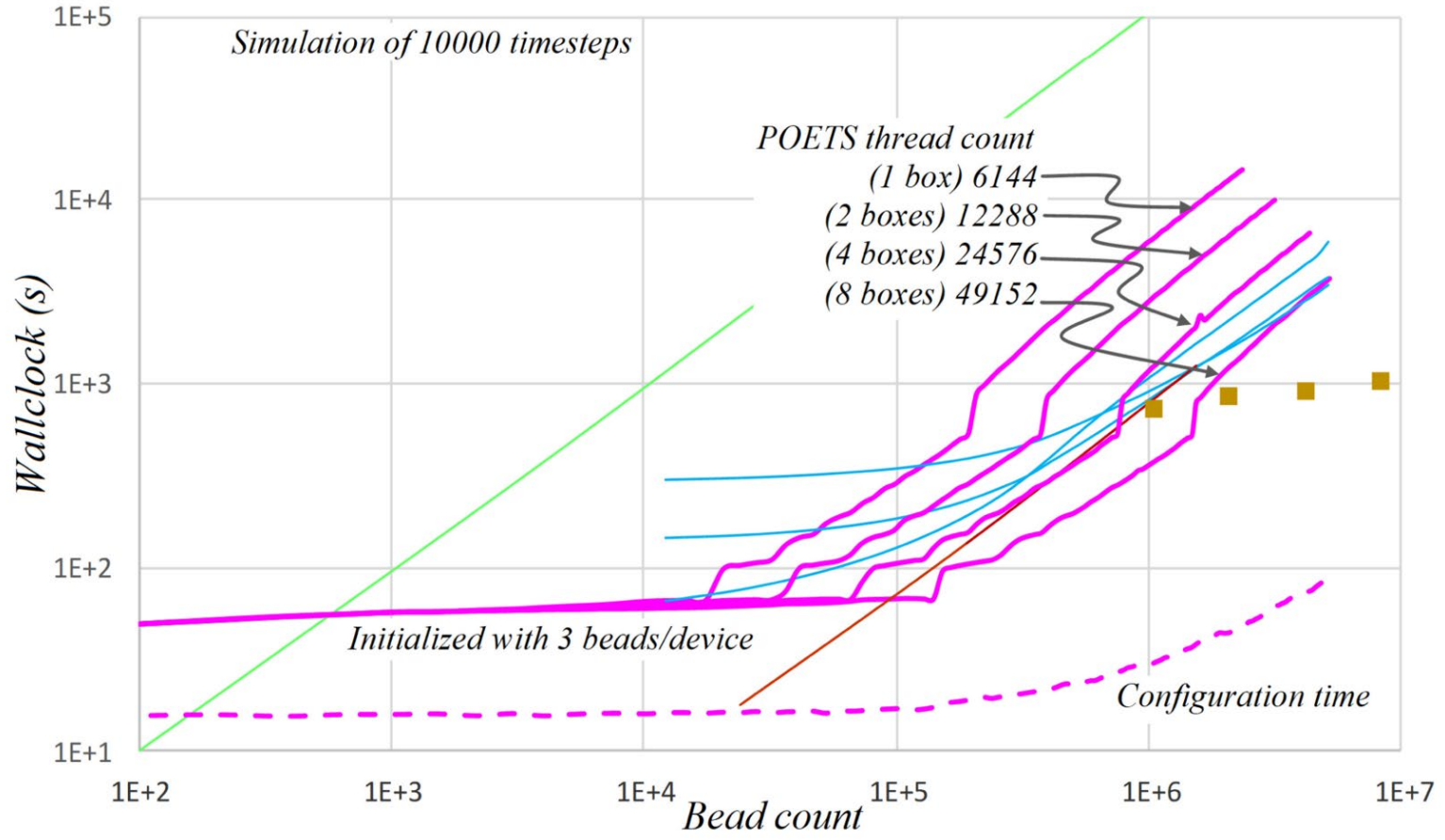
Reliable inter-FPGA links: N, S, E and W

2 × DDR3 DRAM and 4 × QDRII+ SRAM in total



Separate message and memory NoCs reduce **congestion** and avoid message-dependant **deadlock**

POETS Performance: Dissipative Particle Dynamics



POETS: The Good, the Bad and the Ugly

Strengths:

- Complete application + toolchain + hardware stack
- Large-scale hardware evaluation platform
- It worked!
- Good performance in many application domains
  - It exceeded standard performance in *some* domains
- Many untapped opportunities

Weaknesses:

- Raw hardware performance
  - We had to use old FPGAs : 10 years old when we started in 2016
  - Never completed roll-out to new FPGAs : covid supply-chain fun
  - GPUs and multi-core chips are just newer and faster
- Lack of a clear “best” hardware configuration to harden to ASIC
  - Different applications needed different parameterisations of Tinsel

SONNETS: The Next 5 Years

- EPSRC £7M Programme Grant
  - Started on 1st of Jan 2024
- Three university partners:
  - Imperial
  - Newcastle
  - Southampton
- Numerous industrial and academic collaborators

University of Southampton

SONNETS: The research vision

We will address three inter-related challenges:

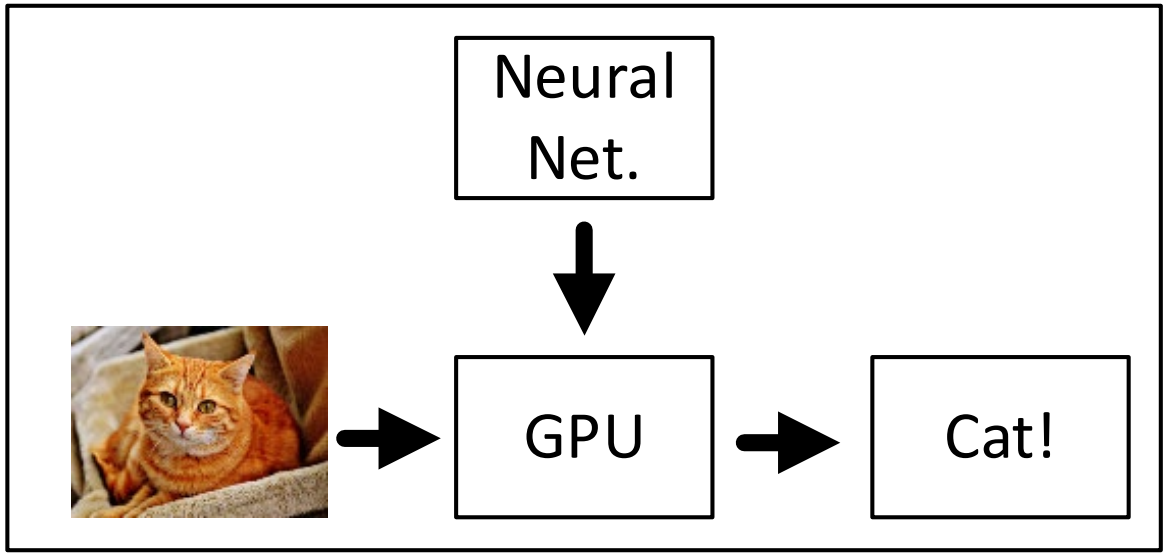
Compute for next-gen AI: what do compute systems for *3rd wave AI* look like, once we move beyond black-box Convolutional Neural Networks and tensors?

Event-triggered computing (ETC): how do we program and manage billions of *heterogeneous processors* spread across the cloud?

Avoiding the next crash: how do you evaluate the *financial risk of the entire UK* financial system; it cost ~£500bn last time – how do we avoid something worse?

SONNETS views these problems as a spectrum: *the solutions support each other*

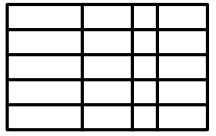
Contemporary Machine Learning

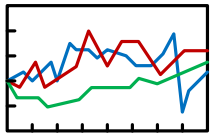


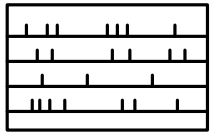
- Regular inputs from single source : *images, audio, text*
- Algorithms optimized for GPUs assumes dense regular inputs
- Black-box algorithms that make simple classifications

Next-generation Machine Learning

- Many input data formats

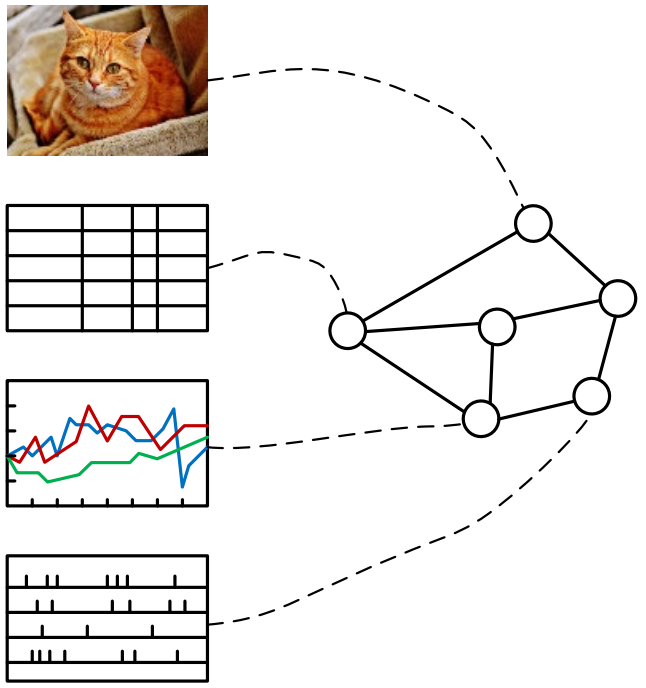






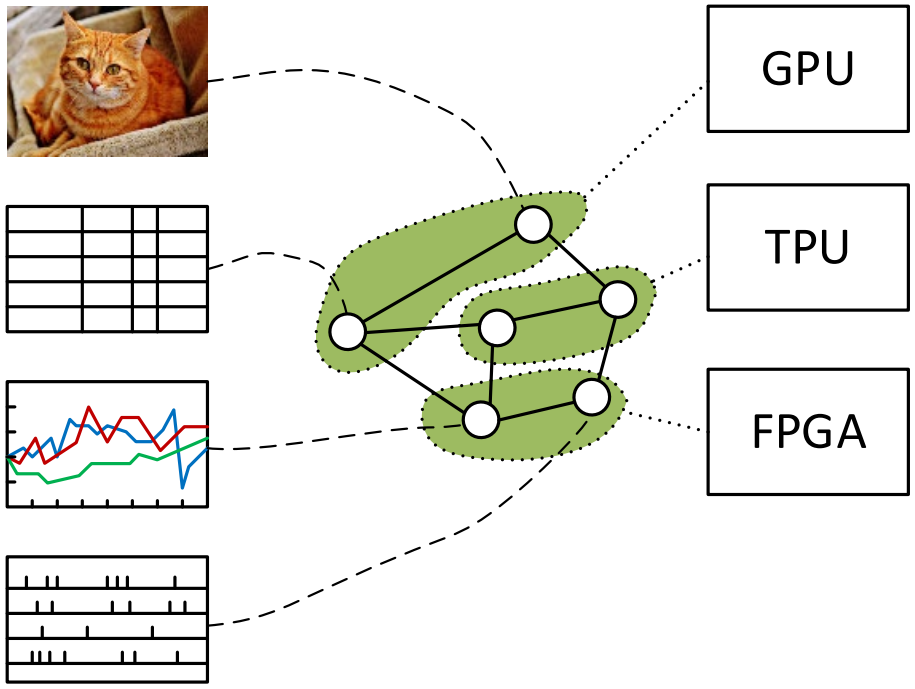
Next-generation Machine Learning

- Many input data formats
- Complex relationships



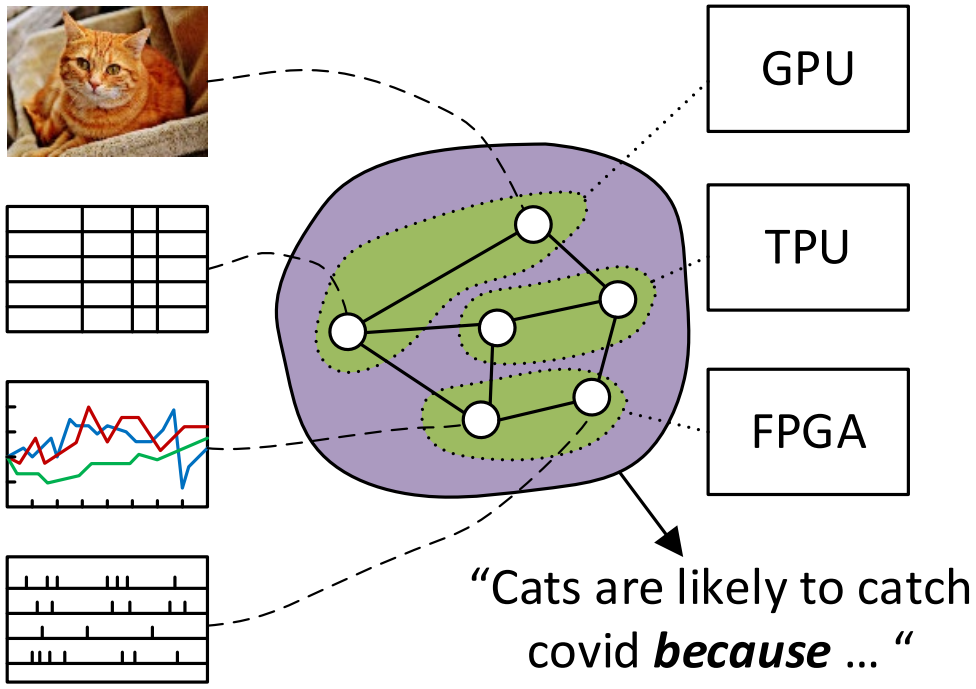
Next-generation Machine Learning

- Many input data formats
- Complex relationships
- Many pools of computation
- Many types of computer



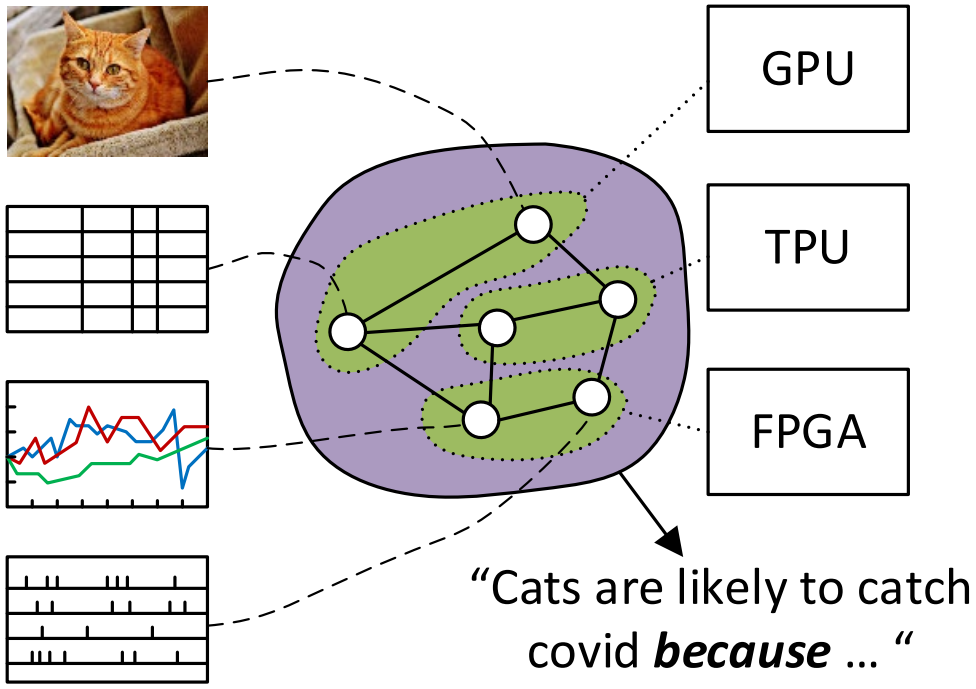
Next-generation Machine Learning

- Many input data formats
- Complex relationships
- Many pools of computation
- Many types of computer
- Complex and nuanced answers
- Explainability



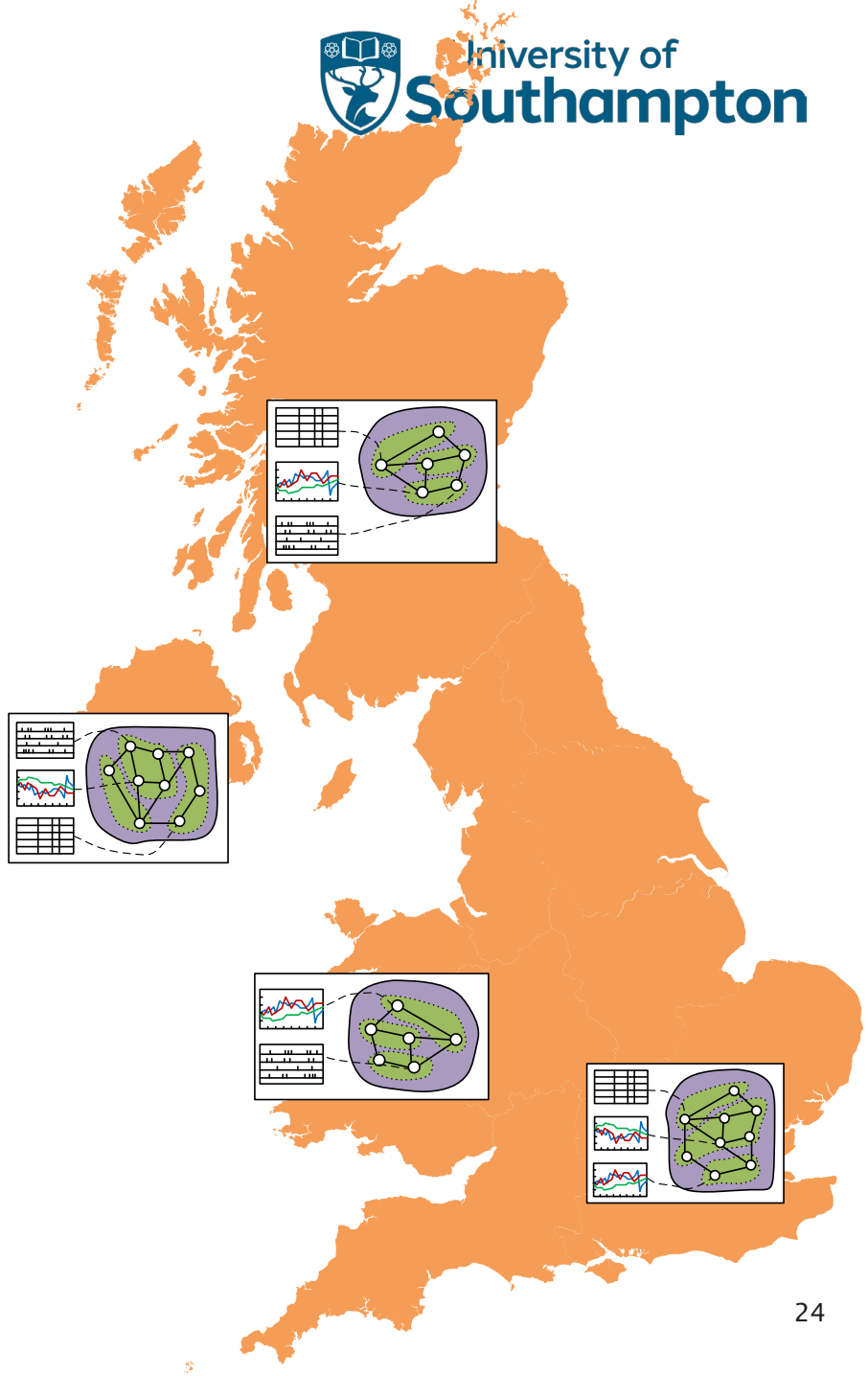
Next-generation Machine Learning

- Many input data formats
- Complex relationships
- Many pools of computation
- Many types of computer
- Complex and nuanced answers
- Explainability
- We can achieve this with Event-Triggered Computing (ETC)



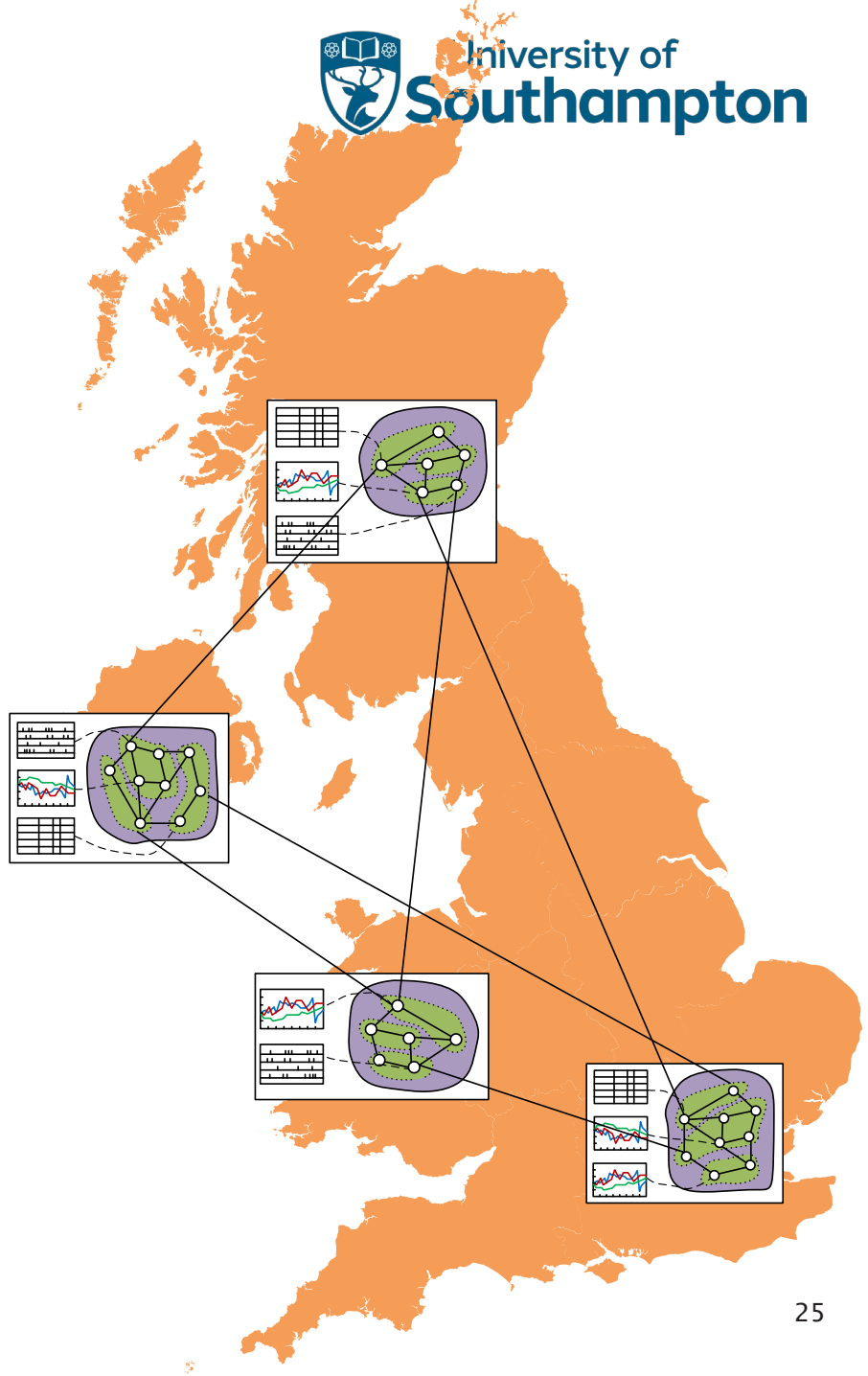
Driver: UK financial risk

- We stress-test institutions individually
  - Ignores inter-dependencies
  - Hides correlations and patterns



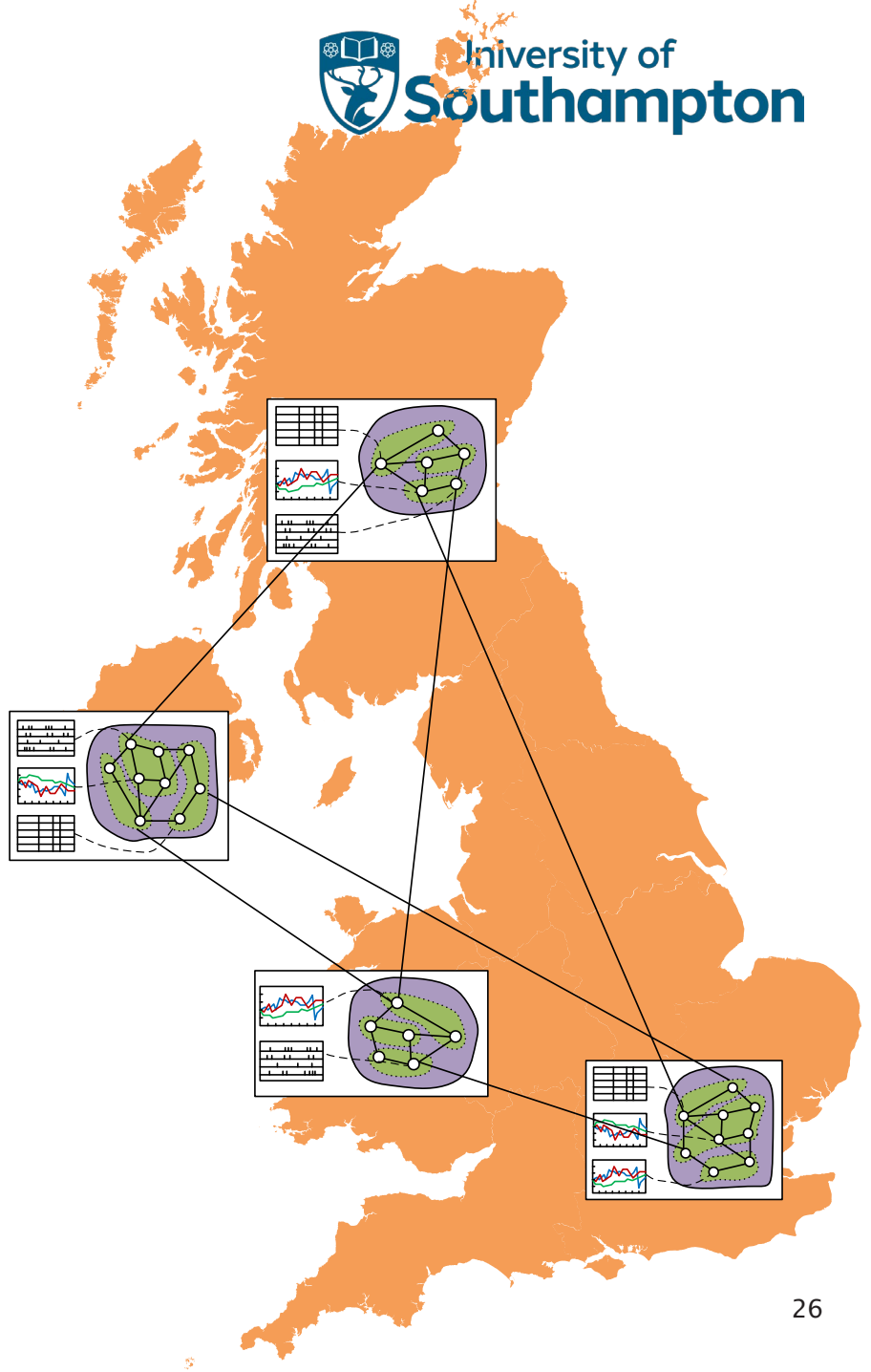
Driver: UK financial risk

- We stress-test institutions individually
  - Ignores inter-dependencies
  - Hides correlations and patterns
- Nation-wide stress-tests are too expensive
  - Must model billions of interacting positions
  - Need data sourcing in each institution



Driver: UK financial risk

- We stress-test institutions individually
  - Ignores inter-dependencies
  - Hides correlations and patterns
- Nation-wide stress-tests are too expensive
  - Must model billions of interacting positions
  - Need data sourcing in each institution
- SONNETS : make use of *ML* and *ETC*
  - Real-time picture of UK risk exposure
  - Use next-gen *ML* to look for patterns
  - Use *ETC* to co-ordinate billions of cores



Timeliness: push and pull factors

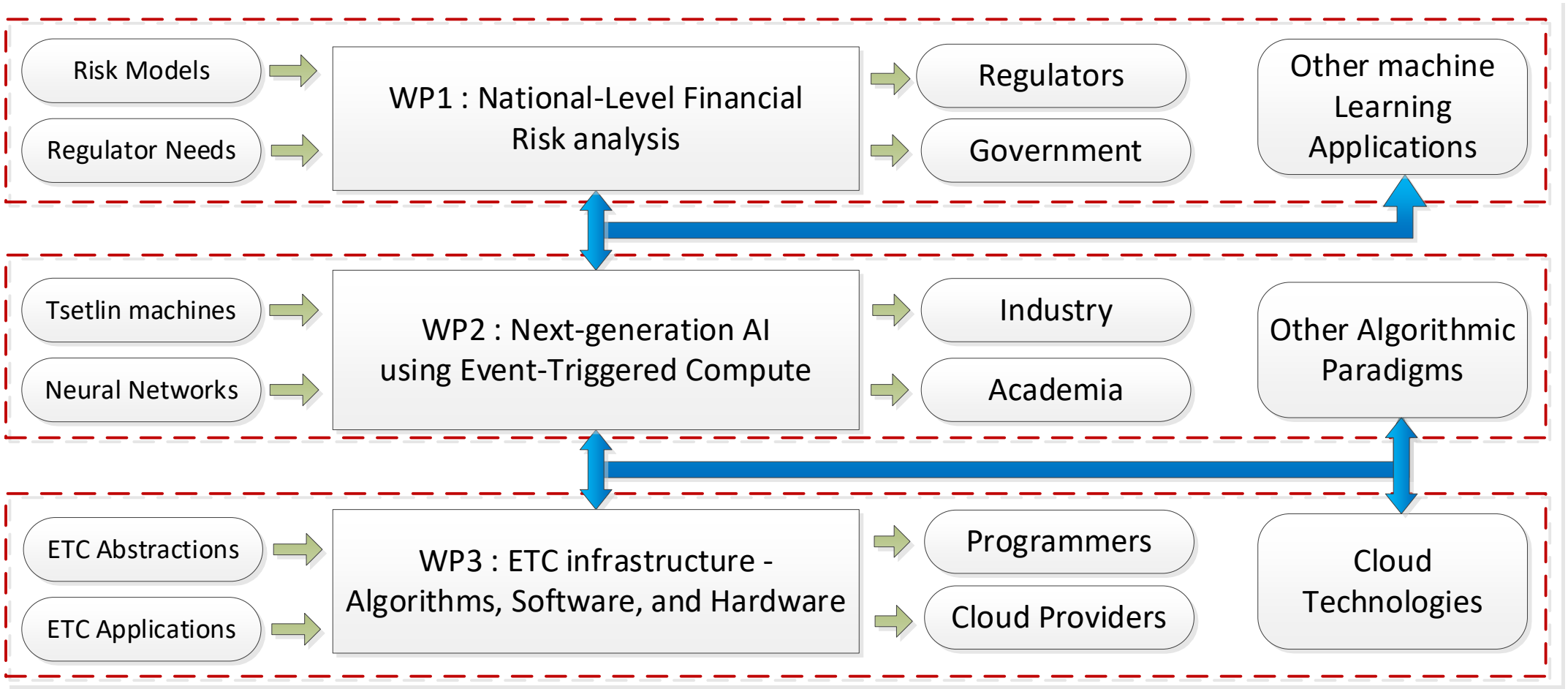
- *Pull factors* : what we need right now
  - **A new direction for ML** : CNNs, GPUs and tensors only get us so far
  - **Nation-level modelling is critical** : risk-analysis, pandemic models, energy, …
  - **Programming capability gap**: software is lagging the hardware we can build

- *Push factors* : what has just become available
  - **Cloud computing** : cost per CPU thread is decreasing
  - **Event-triggered compute** : POETS provides a new programming paradigm

already validated on large-scale computational chemistry

SONNETS bridges the gap between current availability and immediate need

Work-packages



University of Southampton

Selected Challenges related to computation

- Computation problems we want to address
  - Heterogeneity : how do we describe large distributed heterogeneous applications?
  - Verification : how do we prove large distributed heterogeneous apps actually work?
  - Scaling : how do we actually scale the apps up/down and in/out at run-time?
  - Fault-tolerance : how do we keep apps running when nodes or networks fail?
- Some techniques we want to use
  - Event-triggered Computing
  - Formal methods : e.g. Event-B and System refinement
  - Heterogenous cloud compute instances
  - Leverage existing event-streaming and event-broker infrastructure

Collaboration opportunities

What we have: 5 years of funding

- 2.5 FTE/year over 10 academic staff
  - ML, risk analysis, heterog. compute,...
- 6 FTE/year of research fellows
- 9 PhD positions
- A mandate to do blue-sky research

What you have (?)

- Interest in cloud/data-centre/edge
- Interest in ML accelerators (?)
- Optimised building-blocks
- Heterogeneous SoCs and CPUs
- Platforms for scale-out

What we need

- Input on future architectures
- Challenging platforms to program
- Accelerated libraries (e.g. ML)

What you need (?)

- New complex driver applications
- Academics with strange ideas

Conclusion

- Event-triggered Computing is a simple but useful abstraction
  - Has allowed us to develop complex fine-grain asynchronous applications
  - Enabled creation of massively concurrent Tinsel architecture
- With SONNETS we want to tackle large distributed heterogenous problems
  - Real-time national risk analysis: a big problem to drive innovative solutions
  - Machine learning and modelling: complex application domains to work withing
  - New programming paradigms: support design, verification, and execution in the cloud
- We know ***where*** we're going...

  ... but not yet ***how*** we'll get there