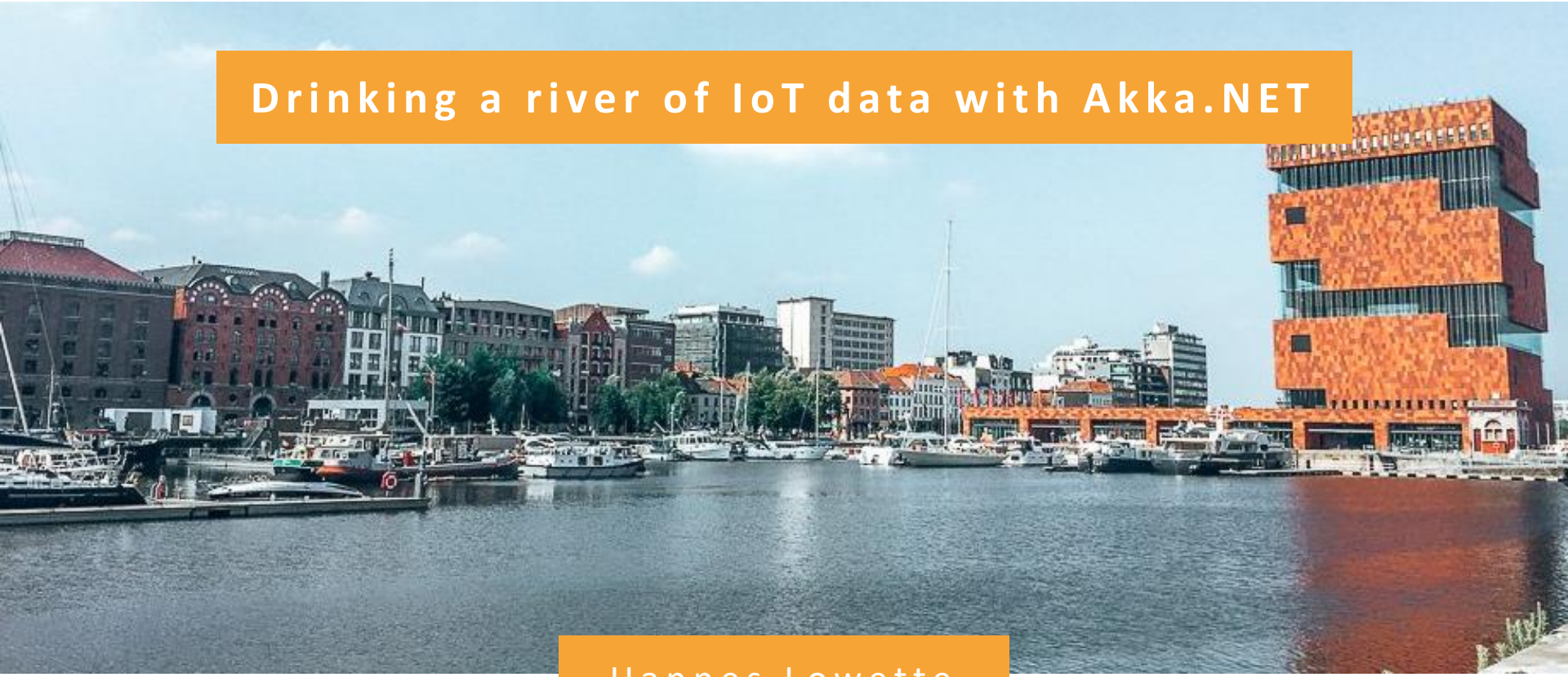
AXXES
IT CONSULTANCY

Drinking a river of IoT data with Akka.NET



Hannes Lowette

IT IS ABOUT PEOPLE

1. Introduction to Akka.NET
2. The problem domain
3. How does Akka.NET fit?
4. Implementation details
5. Beyond this talk …

Introduction to Akka.NET

History & Principles

IT IS ABOUT PEOPLE

AXXES
IT CONSULTANCY

Actor model: the origin

Research paper in 1973:

- Carl Hewitt, Peter Bishop and Richard Steiger
- Designing software inspired by physics
- Aimed at many independent microprocessors

Further refinement:

- Irene Greif → operational semantics for the Actor model
- Henry Baker & Carl Hewitt → axiomatic laws for Actor systems
- William Clinger → denotational semantics based on power domains (1981)
- Gul Agha → transition-based semantic model (1985)

AXXES
IT CONSULTANCY

1980's: Major adoption

Ericsson AXD 301 Telco System:

- Invention of **Erlang**
- Fault-Tolerant
- Distributed
- Concurrent
- 2 million lines of code
- 99,9999999% uptime (9 nines)

(31ms downtime per year)

ERICSSON

AXXES IT CONSULTANCY

Evolution of Akka.NET

- **2010:** Akka on JVM (Scala)
- **2013:**
  - Aaron Stannard: started porting Akka to .NET
  - Roger Johansson: started porting Akka to .NET (Pigeon)
- **2014:**
  - Combined codebases & effort
  - Permission from Lightbend to call it Akka.NET
- **2015:** First version of Akka.NET released
- **Today:** team expanded, more stable & complete, …

AXXES
IT CONSULTANCY

What happened in 2015?

3 Actor frameworks for .NET get released:

- **Feb 2015:** Project Orleans v 1.0.0
- **April 2015:** Akka.NET v 1.0.0
- **April 2015:** Service Fabric Reliable Actors v1.0.x

Orleans

Microsoft Azure
Service Fabric

AXXES
IT CONSULTANCY

Why 2015?

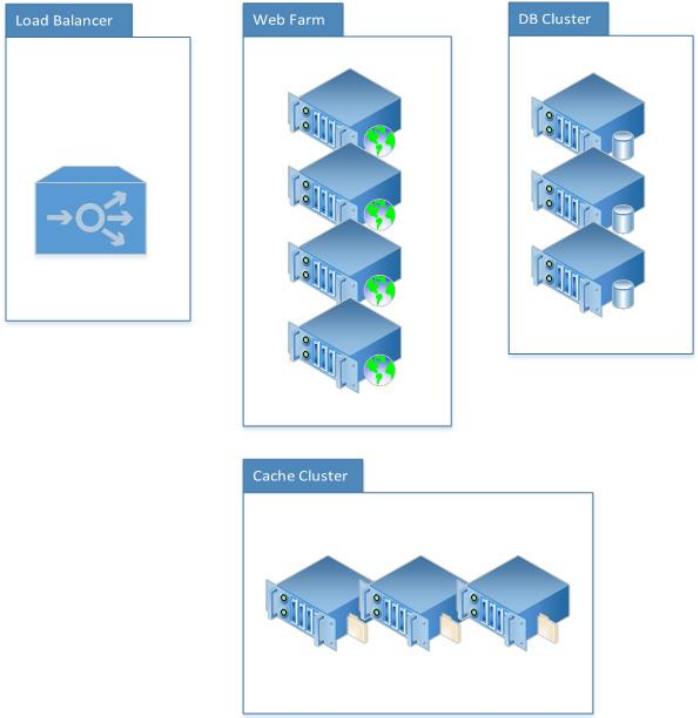
If these concepts exist since the 70's…

AXXES
IT CONSULTANCY

Classic scaling can't keep up

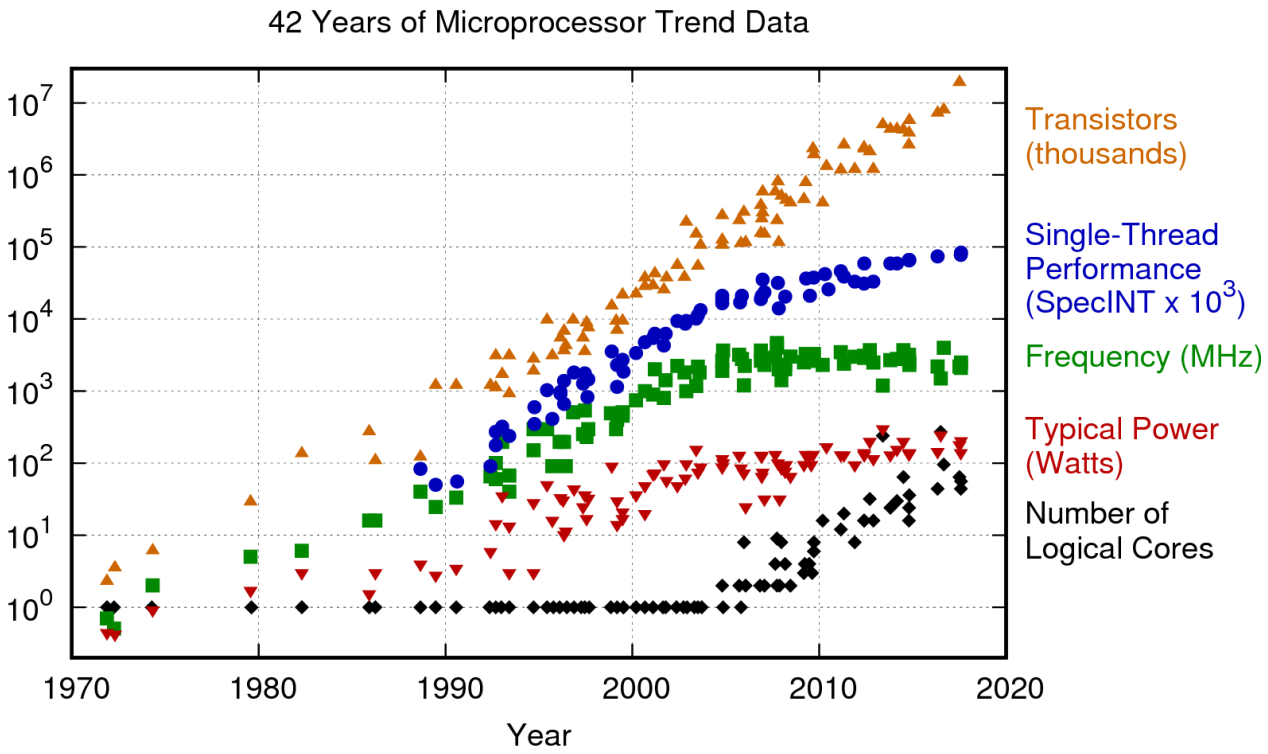
Explosion of # consumers:

- Smartphone apps
- Internet of things
- Regular website traffic



AXXES
IT CONSULTANCY

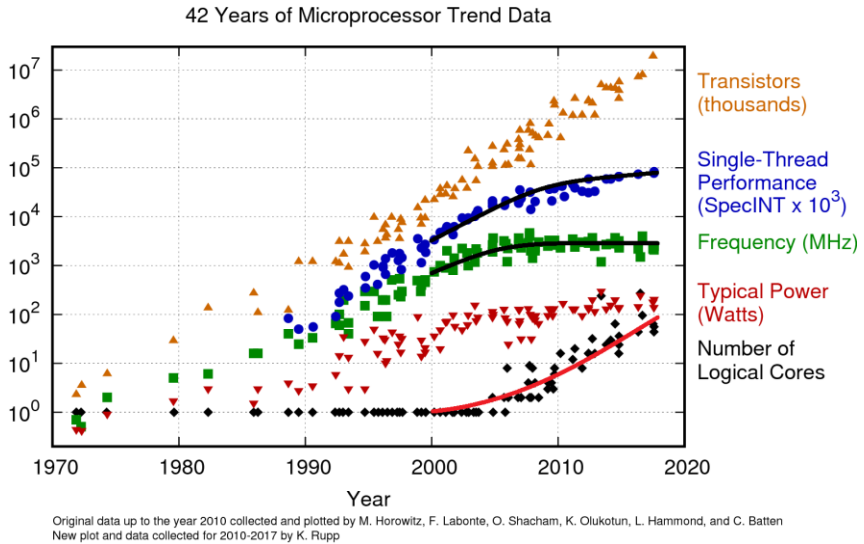
Free lunch was over



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

AXXES
IT CONSULTANCY

Free lunch was over



- Clock frequency has stabilized for over a decade
- Single thread performance growth is slowing down

But …

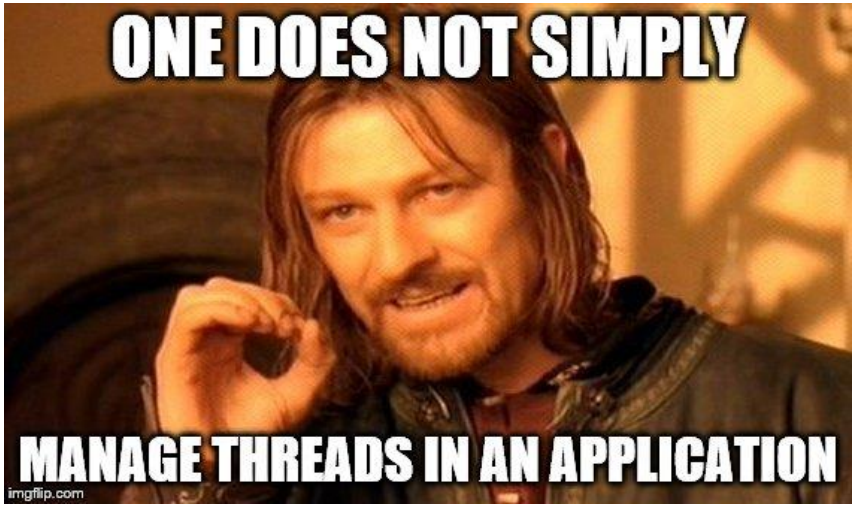
The number of cores is increasing!

AXXES
IT CONSULTANCY

Parallelism is the salvation

Problems of parallel execution:

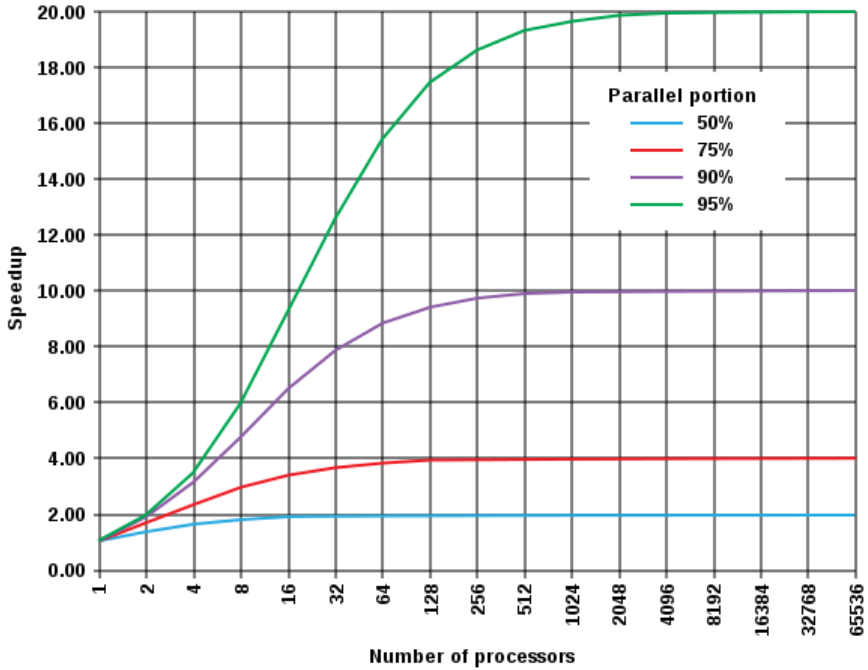
- Shared state:
  - Race conditions
  - Blocking calls
  - Deadlocks
- Some code can't be parallelized



Serialization hurts scalability

AXXES
IT CONSULTANCY

Amdahl's Law



- The theoretical speedup
- For a fixed workload
- When increasing resources
- Depending on how much can be parallelized

90% isn't going to cut it.

AXXES
IT CONSULTANCY

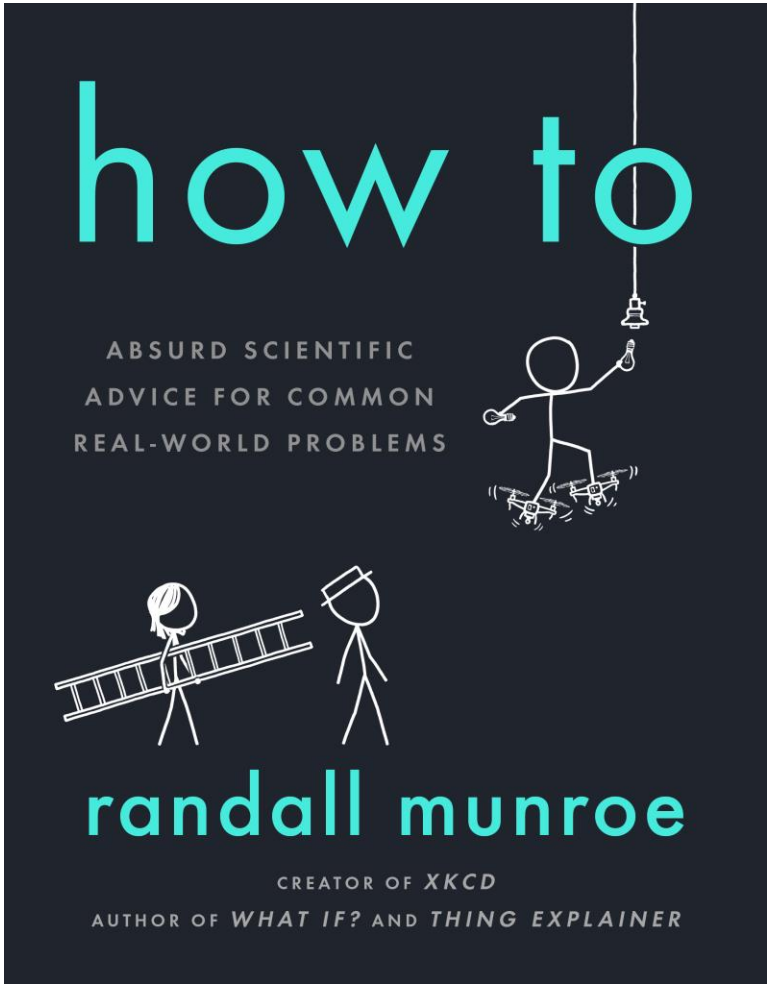
Can actor models help?

Actor models bring:

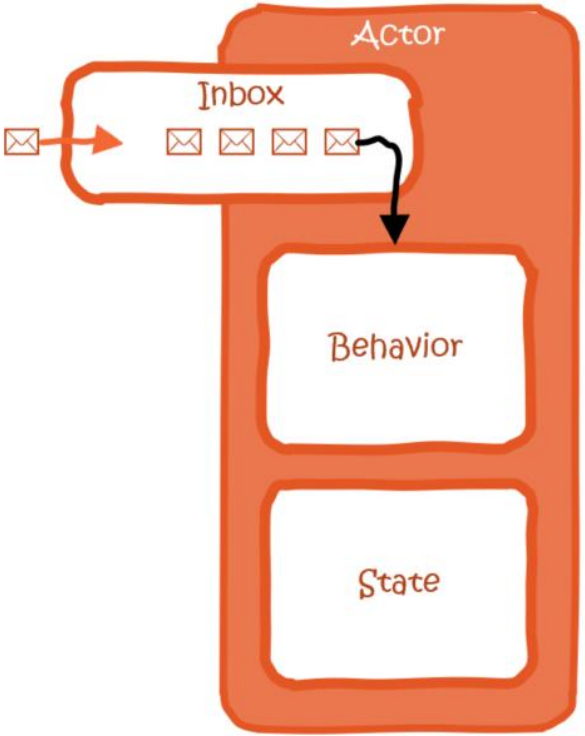
- High degree of parallelization for stateful systems
- Reactive patterns
- Fault tolerance (self healing)

AXXES
IT CONSULTANCY

But, HOW?



The Actor



Simple object

- Holds its own state (no shared state)
- Only input: messages
- Inbox:
  - Processed in order
  - 1 message at a time

→ An Actor runs single threaded

→ no concurrency problems

AXXES
IT CONSULTANCY

The Actor

```
public class MyActor : UntypedActor
{
    protected override void OnReceive(object message)
    {
        if (message is MyMessage myMessage)
            DoSomething(myMessage);
    }

    private void DoSomething(MyMessage myMessage)
    {
        // TODO: handle the message here
    }
}
```

AXXES
IT CONSULTANCY

Messages

- Simple objects
- Immutable!
  - Akka.NET doesn't enforce this
  - You SHOULD NOT try to exploit this
- Might cross machine boundaries
- Throughput:
  - Claimed: 50 M/s on a single machine
  - Well over 1 M/s on my laptop

AXXES
IT CONSULTANCY

Messages

```
public class MyMessage
{
    public int IntProperty { get; }
    public string StringProperty { get; }
    public ImmutableArray<decimal> Values { get; }

    public MyMessage(int intProperty, string stringProperty, ImmutableArray<decimal> values)
    {
        IntProperty = intProperty;
        StringProperty = stringProperty;
        Values = values;
    }
}
```

AXXES
IT CONSULTANCY

The Actor System

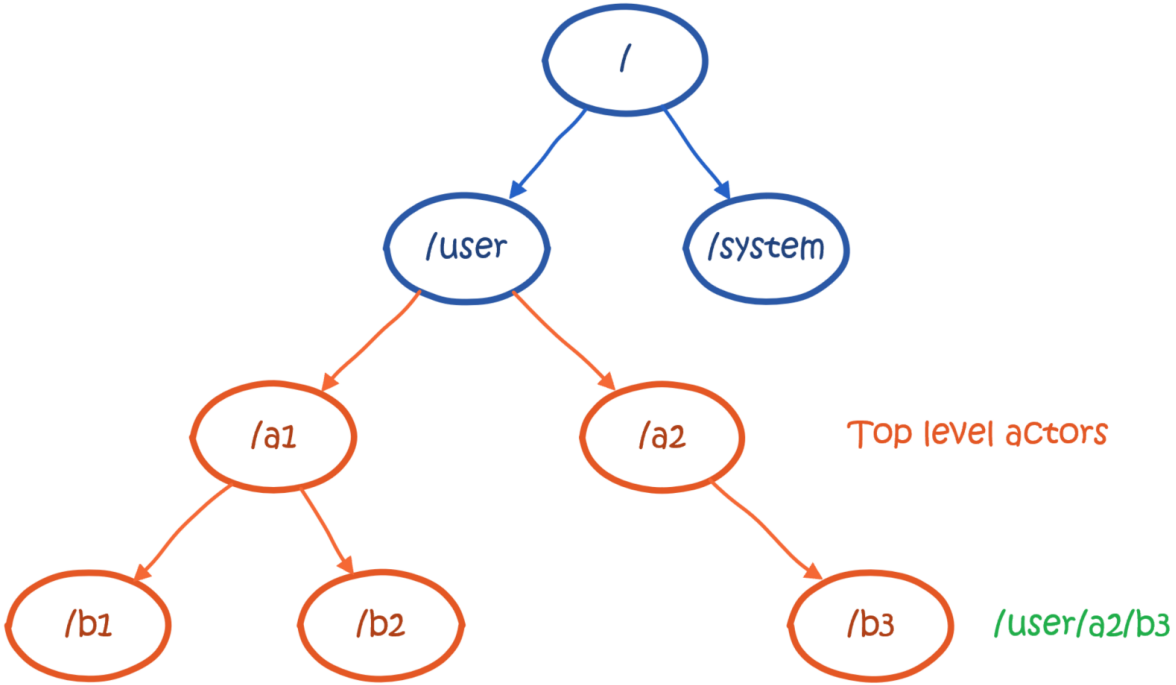
- Manages actor life cycles
- Manages messaging
- Manages inboxes
- Manages thread scheduling
- Manages the system event bus
- …

AXXES
IT CONSULTANCY

The Actor System

```
public static void Main(string[] args)
{
    var actorSystem = ActorSystem.Create("MyActorSystem");

    var myActorProps = Props.Create<MyActor>();

    // Creates /user/my-actor-name
    var myActor = actorSystem.ActorOf(myActorProps, "my-actor-name");
  IActorRef
    var message = new MyMessage(1, "Hello", ImmutableArray<decimal>.Empty);

    myActor.Tell(message);
}
```

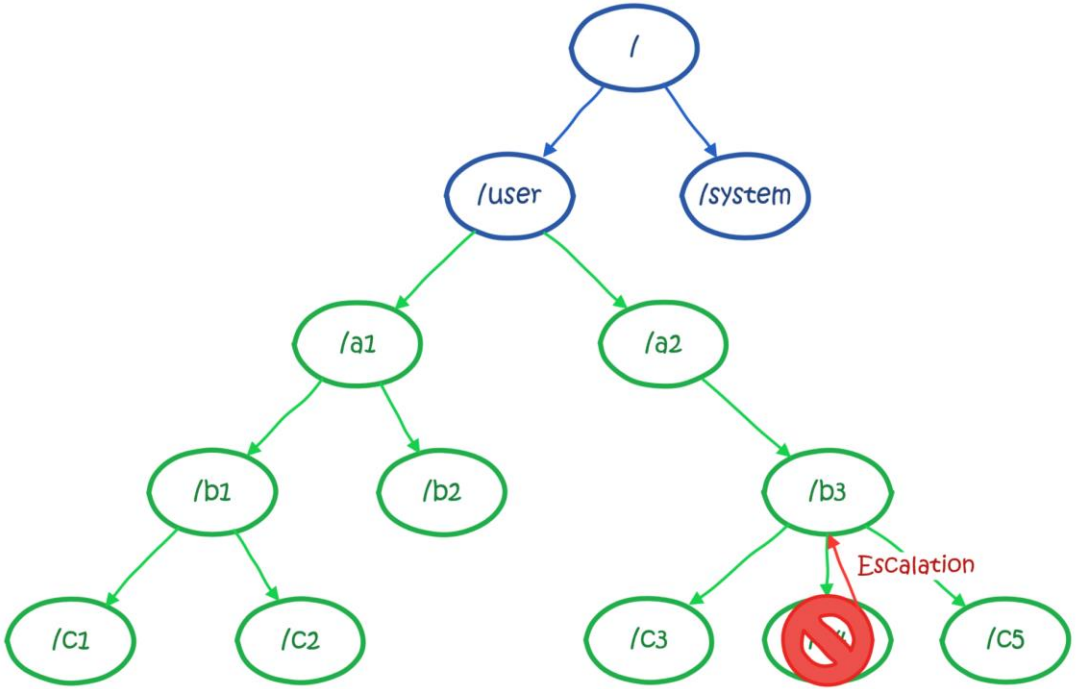
The actor hierarchy



- Actors can have children
- Position in the hierarchy = address
- Directly under /user = ‘top level actors’
- 3 default actors:
  - /
  - /user
  - /system

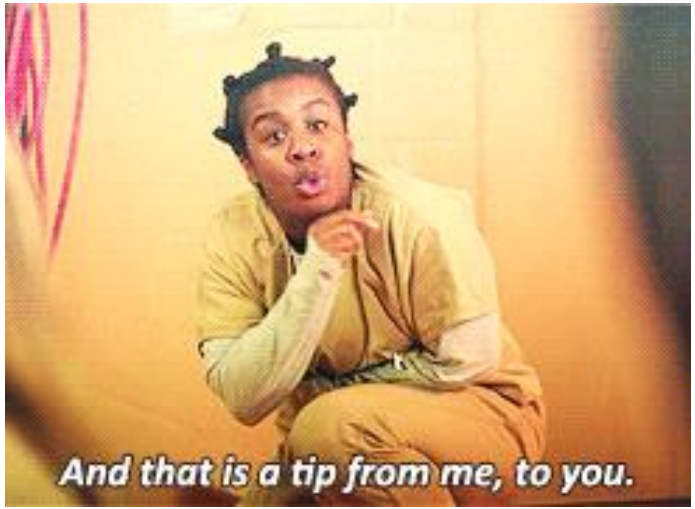
Fault Tolerance - Supervision

- Errors are escalated to the parent
- Parent decides OR escalates further
- Strategy:
  - OneForOne: only the failing actor
  - OneForAll: all children
- Action:
  - Resume
  - Stop
  - Restart



AXXES
IT CONSULTANCY

General development ideas

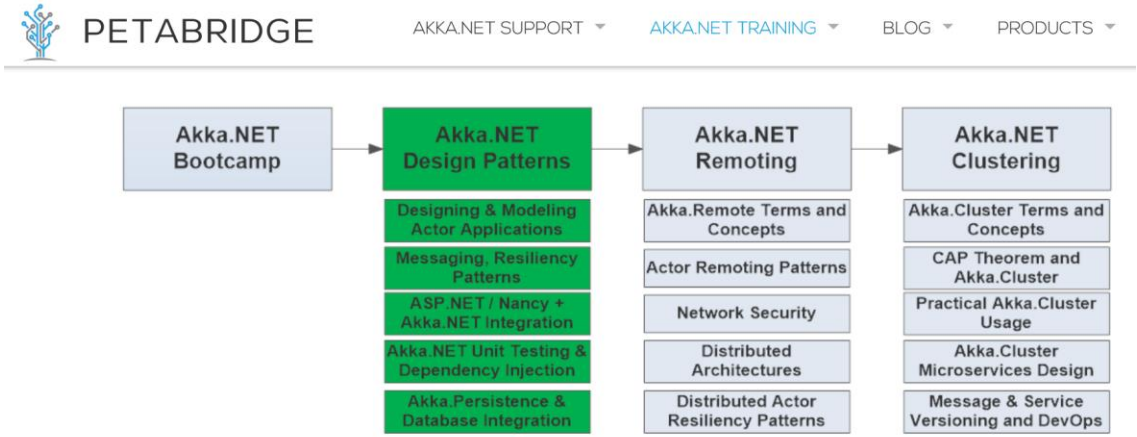


- Split workloads into small chunks
- Make separate actors for every task
- Push risk to the edges,
  handle faults there
- Avoid ‘bottleneck actors’

AXXES
IT CONSULTANCY

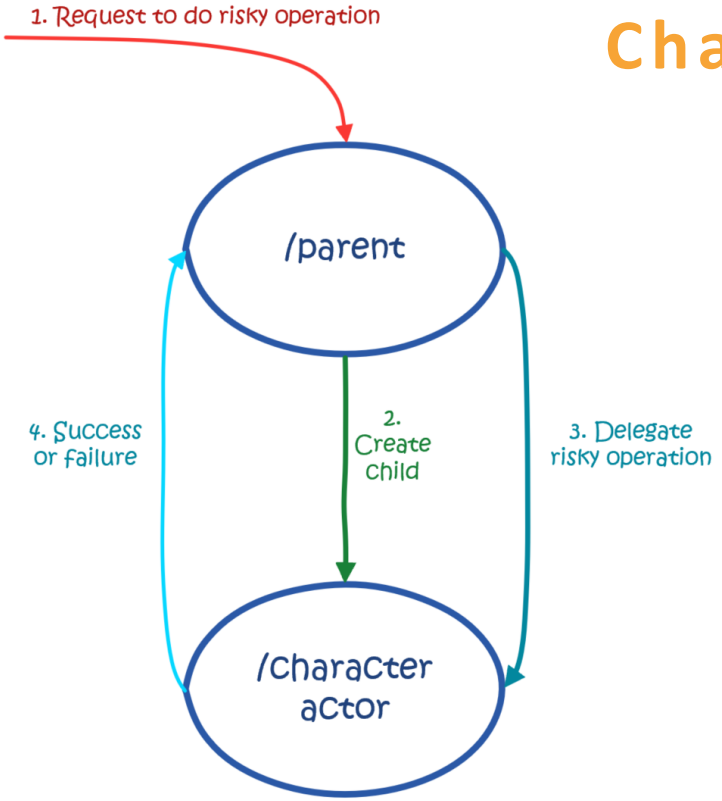
New design patterns

- Fan-out Pattern
- Parent Proxy Pattern
- Consensus Pattern
- Character Actor
- …



AXXES
IT CONSULTANCY

Character Actor



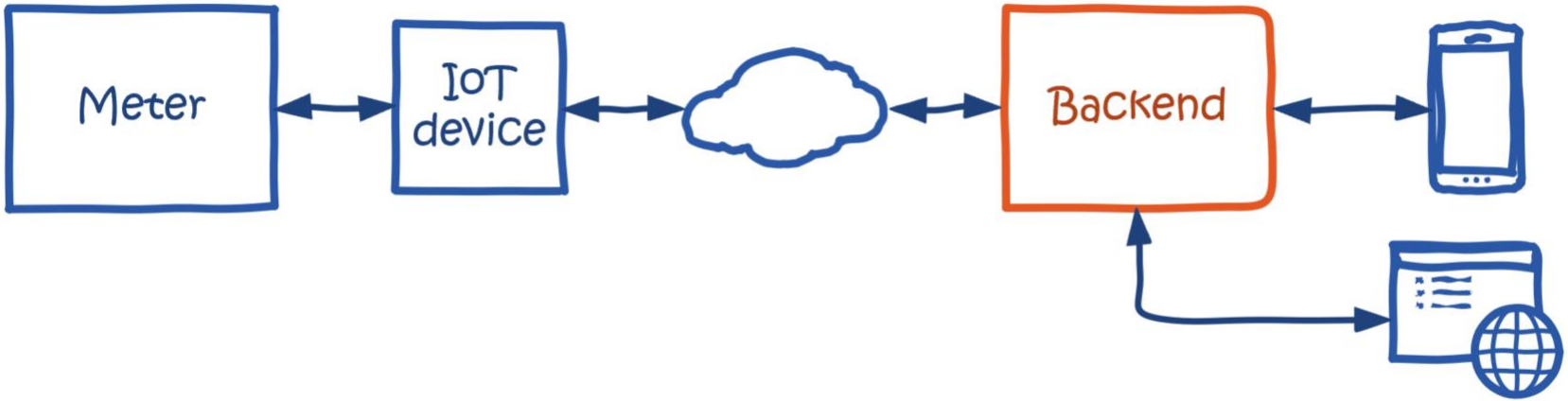
The problem domain

What are we going to solve?

IT IS ABOUT PEOPLE

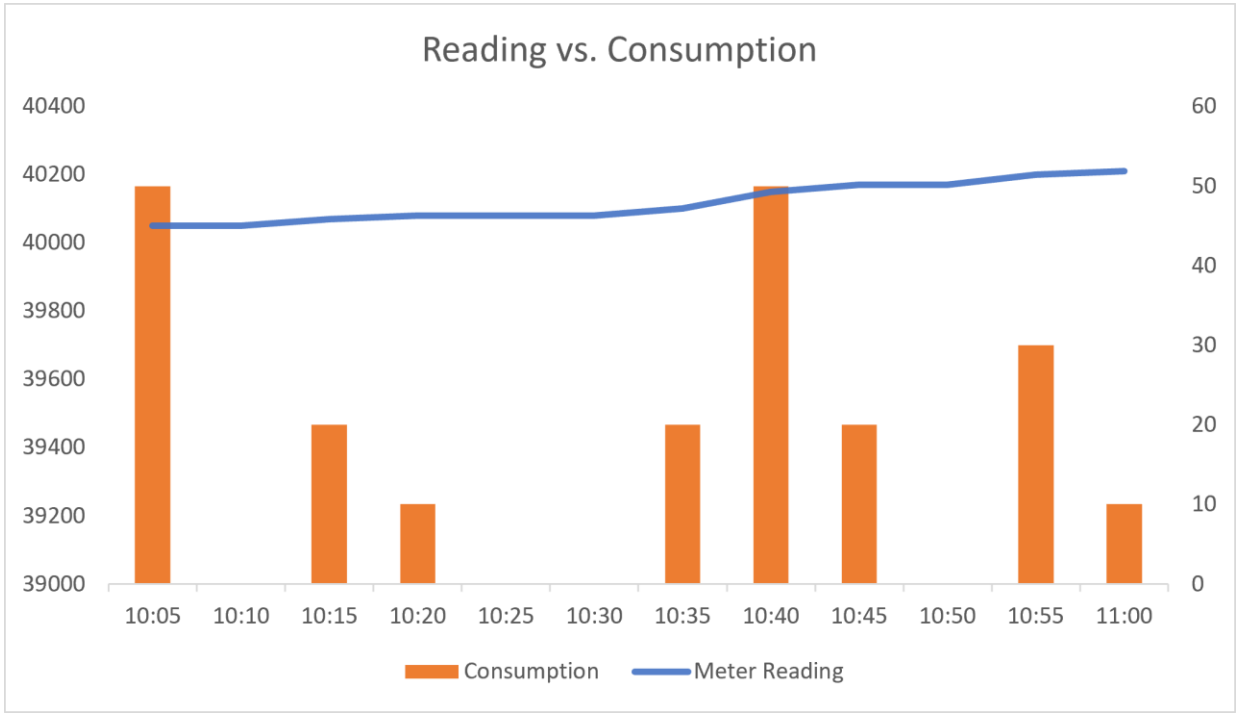
AXXES
IT CONSULTANCY

Connection situation



AXXES
IT CONSULTANCY

Reading vs Consumption



AXXES
IT CONSULTANCY

What are we interested in?

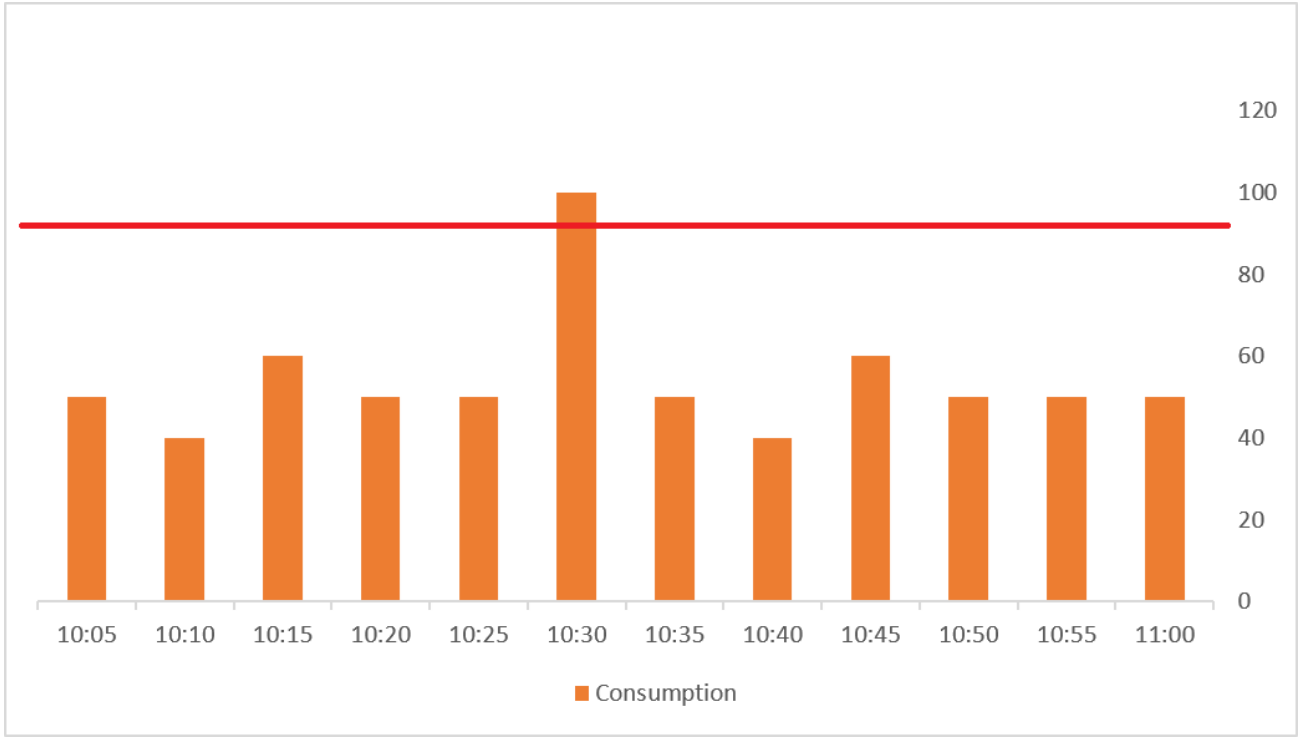
Storage of historic usage

- Storage of (normalized) values
- Plotting of consumption graphs
- Comparison of time periods

Alerting

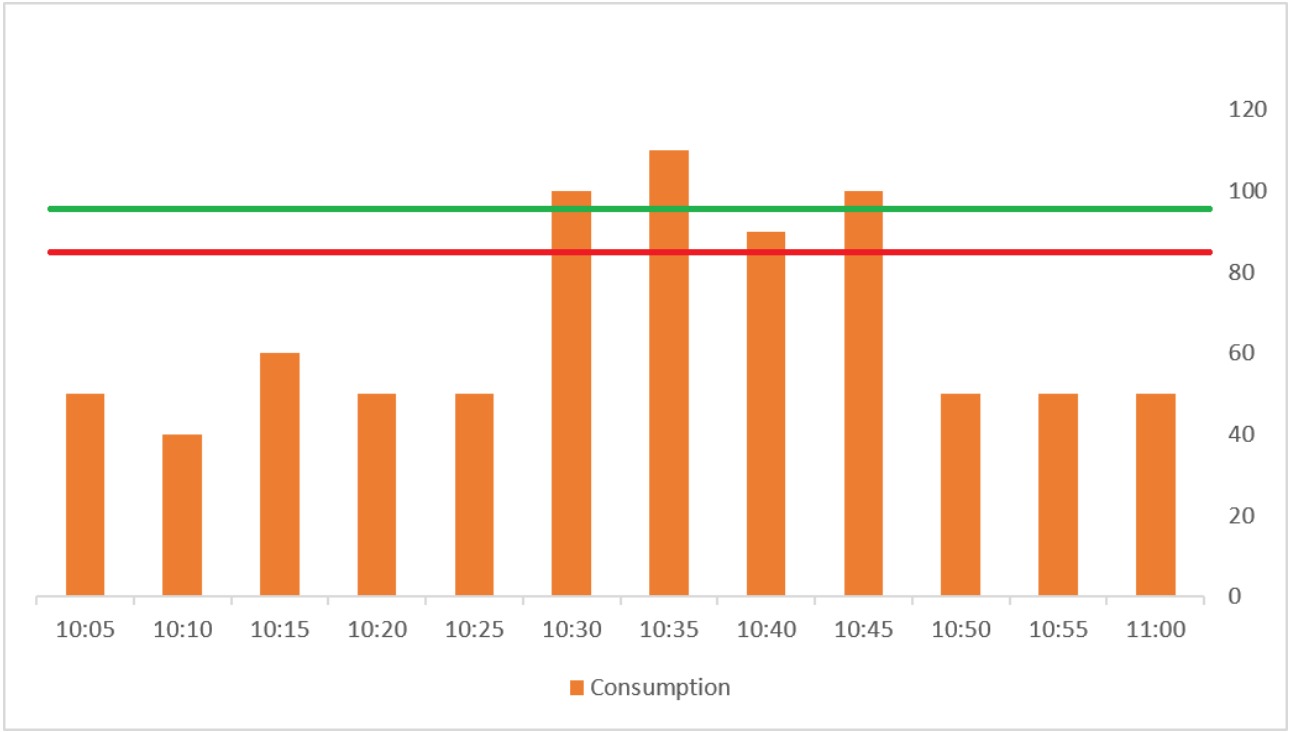
- When momentary consumption exceeds threshold
- When a threshold is exceeded for a period

Momentary threshold



AXXES
IT CONSULTANCY

Periodic threshold



Periodic threshold

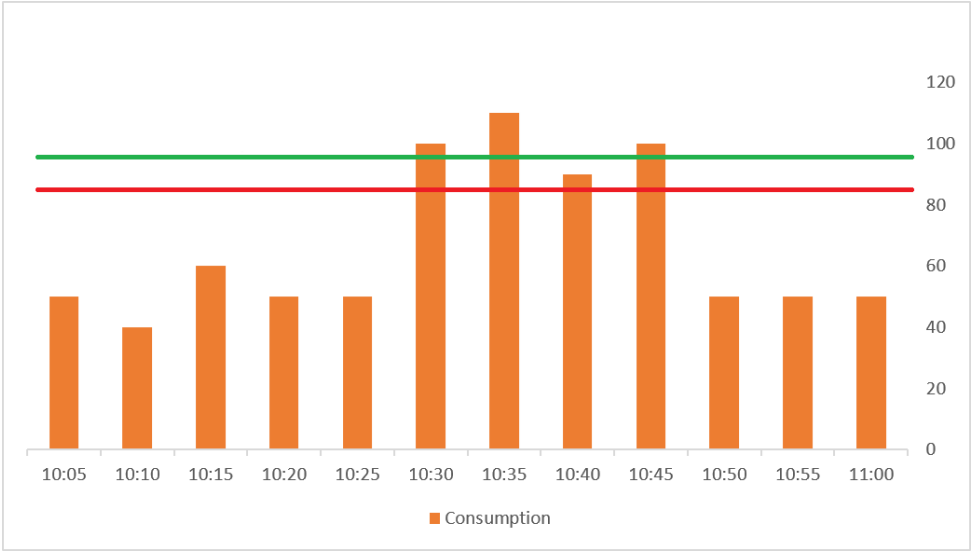
Alert after 20 minutes

Green line:

- Exceeded on average
- Not exceeded every time block

Red line:

- Exceeded on average
- Exceeded every time block

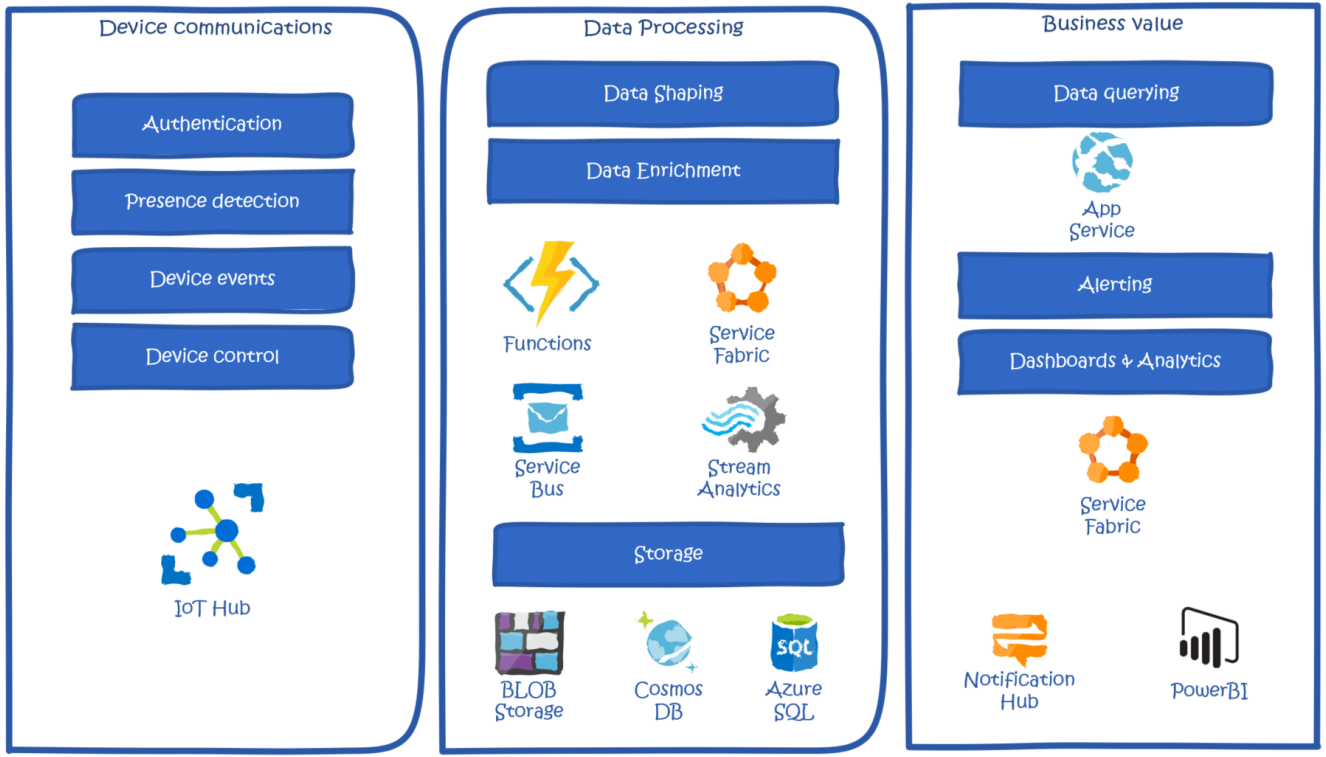


How does Akka.NET fit?

What part of the solution can Akka.NET provide?

IT IS ABOUT PEOPLE

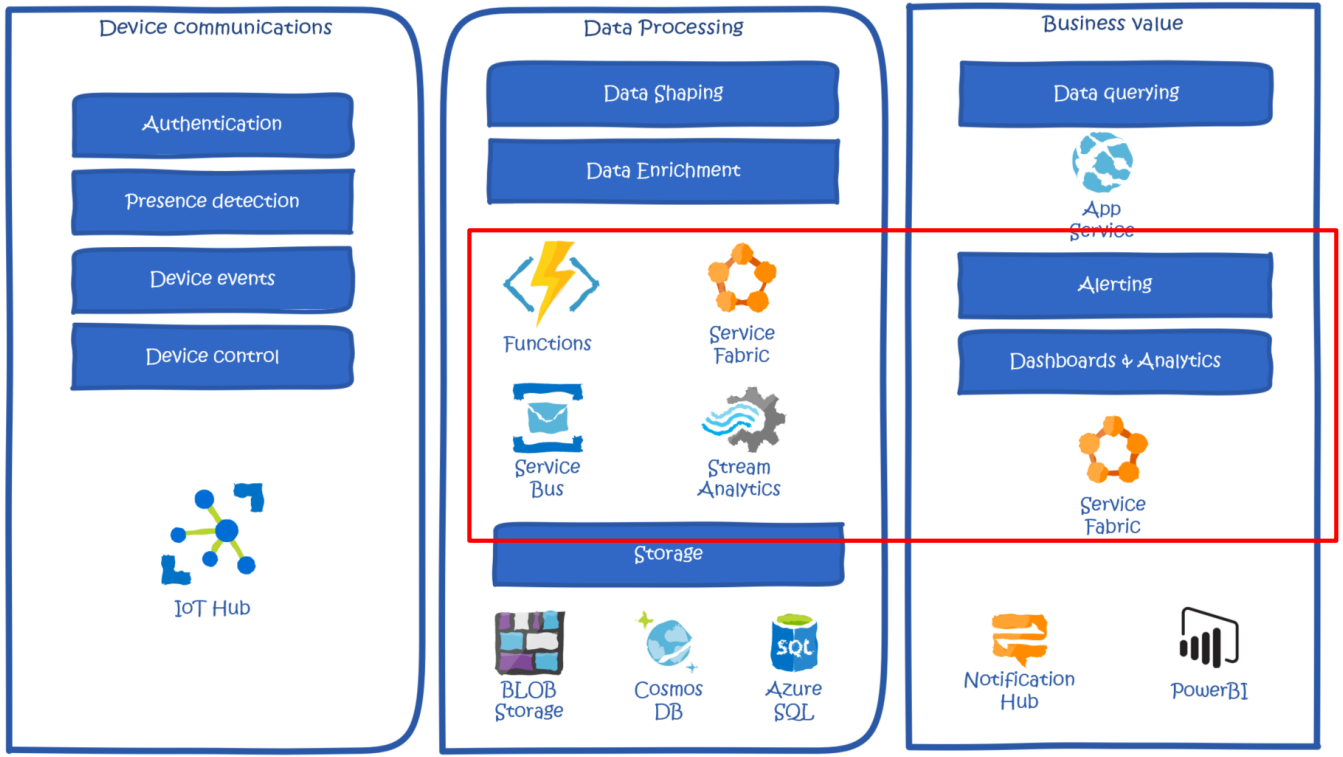
Your typical IoT stack*



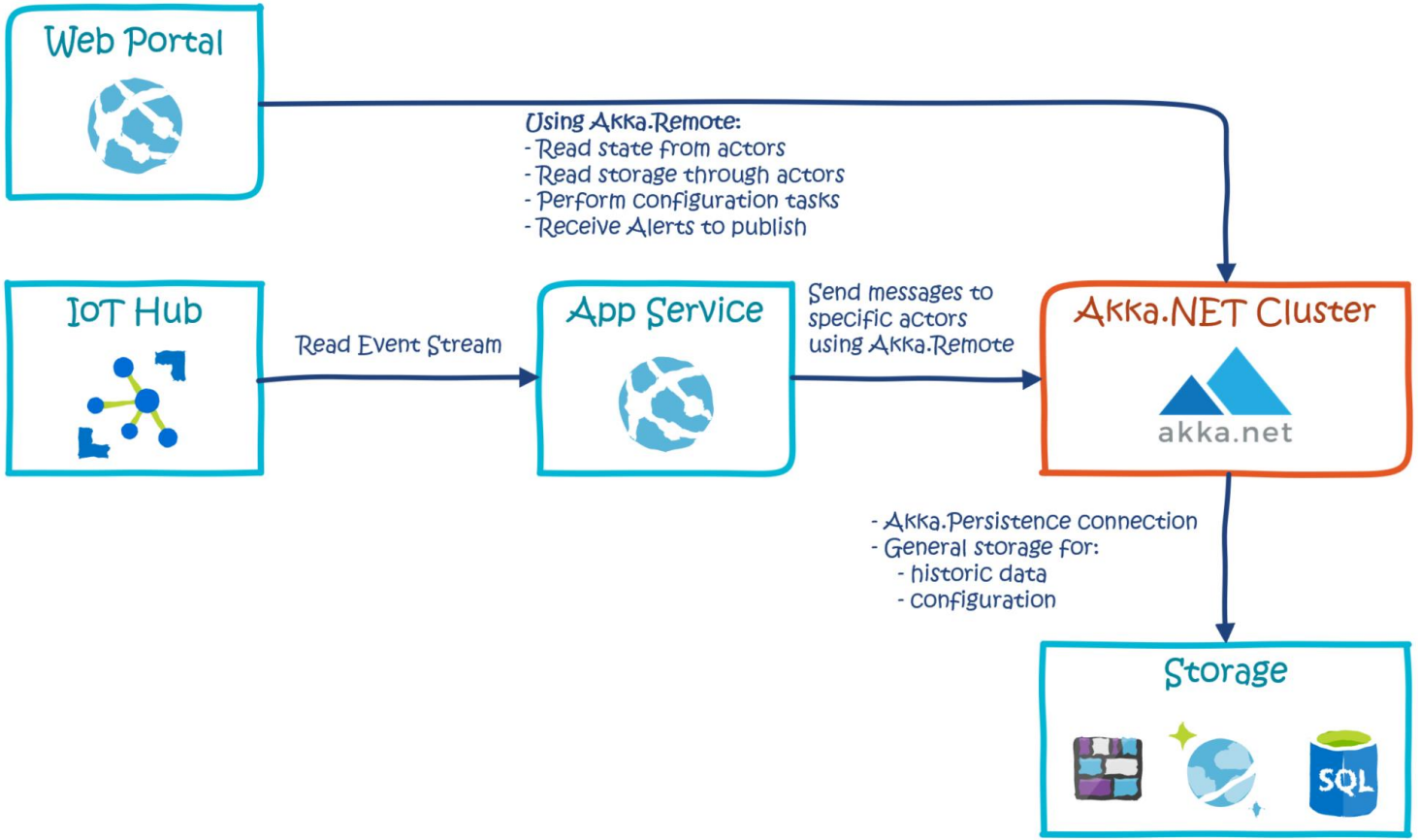
* With Microsoft Azure service examples

AXXES
IT CONSULTANCY

What can we use Akka.NET for?



akka.net


Web Portal
Using Akka.Remote:
- Read state from actors
- Read storage through actors
- Perform configuration tasks
- Receive Alerts to publish
IoT Hub
Read Event Stream
App Service
Send messages to specific actors using Akka.Remote
Akka.NET Cluster
akka.net
- Akka.Persistence connection
- General storage for:
- historic data
- configuration
Storage
SQL

AXXES
IT CONSULTANCY

Don’t be a magpie!

Good fits:

- Gaming backends
- Trading systems
- Internet of Things
- Parallelizable calculations
- … any stateful high throughput application

It doesn’t have to be the whole solution!

Implementation details

How can you use Akka.NET in this scenario?

IT IS ABOUT PEOPLE

IT IS ABOUT PEOPLE

1. Getting messages to the ActorSystem
2. Normalizing measurements
3. Persisting Data
4. Restart behavior

AXXES
IT CONSULTANCY

Getting messages to the ActorSystem

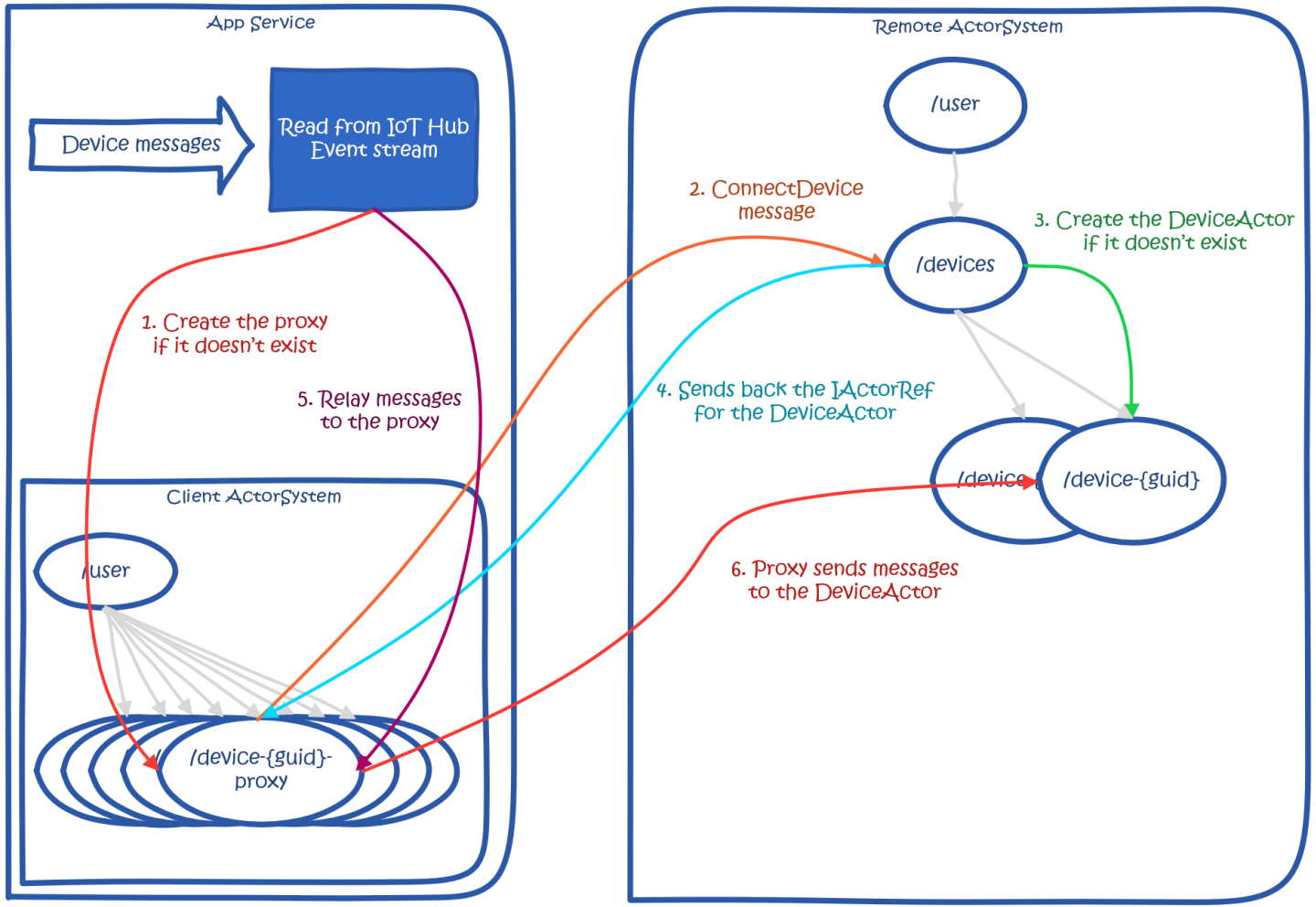
How to use Remoting & Proxy actors

AXXES
IT CONSULTANCY

Akka.Remote

ActorSystems can talk to other ActorSystems

- Remote addressing
- Remote deployment
- Remote messaging
- Location Transparency
- Multiple transports


App Service
Device messages
Read from IoT Hub Event stream
1. Create the proxy if it doesn't exist
5. Relay messages to the proxy
Client ActorSystem
/user
/device-{guid}-proxy
Remote ActorSystem
/user
2. ConnectDevice message
3. Create the DeviceActor if it doesn't exist
/devices
4. Sends back the IActorRef for the DeviceActor
/device
/device-{guid}
6. Proxy sends messages to the DeviceActor

DeviceActorProxy

```
class DeviceActorProxy : ReceiveActor
{
    private readonly Guid _deviceId;
    private IActorRef _deviceActor;

    public DeviceActorProxy(Guid deviceId)...

    protected override void PreStart()
    {
        var devicesActorPath = $"{Constants.RemoteActorSystemAddress}/user/devices";
        var devicesActor = Context.ActorSelection(devicesActorPath);
        var request = new ConnectDevice(_deviceId);
        devicesActor.Tell(request);
    }

    private void HandleDeviceConnected(DeviceConnected message)
    {
        _deviceActor = message.DeviceRef;
    }
```

Runs BEFORE the Actor starts to accept messages.

IActorSelection

Creates an ActorSelection to the DevicesActor in the REMOTE ActorSystem

Sends the 'connect' message to the DevicesActor

When the DevicesActor responds, we will have the IActorRef to the DeviceActor

AXXES IT CONSULTANCY

DeviceActorProxy

```
class DeviceActorProxy : ReceiveActor
{
    private readonly Guid _deviceId;
    private IActorRef _deviceActor;

    public DeviceActorProxy(Guid deviceId)
    {
        _deviceId = deviceId;
        Receive<MeterReadingReceived>(HandleMeterReadingReceived);
        Receive<DeviceConnected>(HandleDeviceConnected);
    }

    protected override void PreStart()...

    private void HandleDeviceConnected(DeviceConnected message)...

    private void HandleMeterReadingReceived(MeterReadingReceived message)
    {
        _deviceActor?.Tell(message);
    }

    public static Props CreateProps(Guid deviceId)
    {
        return Props.Create<DeviceActorProxy>(deviceId);
    }
}
```

More robust than an UntypedActor

This is how you configure message types

Just forward to the actual DeviceActor

This makes refactoring SO MUCH easier

AXXES IT CONSULTANCY

DevicesActor

```
private void HandleConnectDevice(ConnectDevice request)
{
    if (!_deviceActors.ContainsKey(request.Id))
    {
        CreateDeviceActor(request.Id);
    }
    var response = new DeviceConnected(_deviceActors[request.Id]);
    Sender.Tell(response);
}

private void CreateDeviceActor(Guid deviceId)
{
    var props = DeviceActor.CreateProps(deviceId);
    var name = $"device-{deviceId}";
    var deviceActorRef = Context.ActorOf(props, name);

    _deviceActors[deviceId] = deviceActorRef;
}
```

Check if the device has already been created

Sender holds the ActorRef to the sender of the current message

Addresses need to be unique.

Use the Context to create child Actors.

AXXES
IT CONSULTANCY

Normalizing measurements

Making sure downstream actors get consistent data

AXXES
IT CONSULTANCY

Why Normalization?

Writing business logic is easier when you know values are consistent:

- Consistent timestamps
- No gaps
- Incorrect values filtered
- …

Deal with it in one place

AXXES
IT CONSULTANCY

Timestamp correction & buckets

RAW			NORMALIZED		
Timestamp	Reading	Consumption	Timestamp	Reading	Consumption
9:59:25	40000				
10:00:25	40060	60	10:00:00	40035	
10:01:25	40120	60			
10:02:25	40180	60			
10:03:25	40240	60			
10:04:25	40300	60			
10:05:25	40360	60	10:05:00	40335	300
10:06:25	40420	60			
10:07:25	40480	60			
10:08:25	40540	60			
10:09:25	40600	60			
10:10:25	40660	60	10:10:00	40635	300
10:11:25	40720	60			
10:12:25	40780	60			
10:13:25	40840	60			
10:14:25	40900	60			
10:15:25	40960	60	10:15:00	40935	300

Gap filling

- Do we want to fill this gap?
- If so, how?
- Do other Actors need to know?
  If yes, add a flag to the message

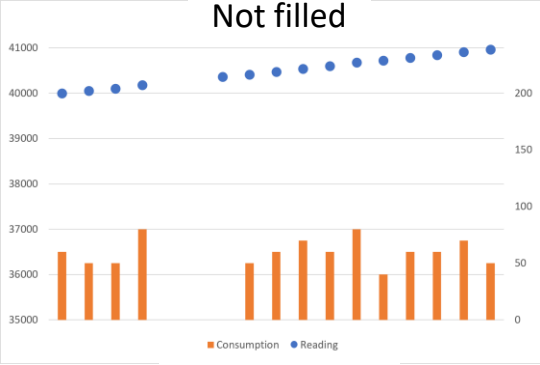
There is no ‘right’ answer

Just one that fits your case

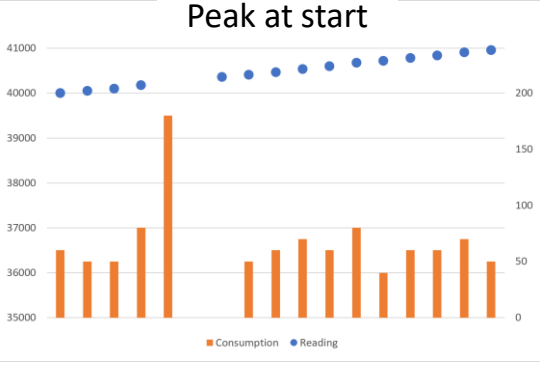
AXXES
IT CONSULTANCY

Possible filling solutions

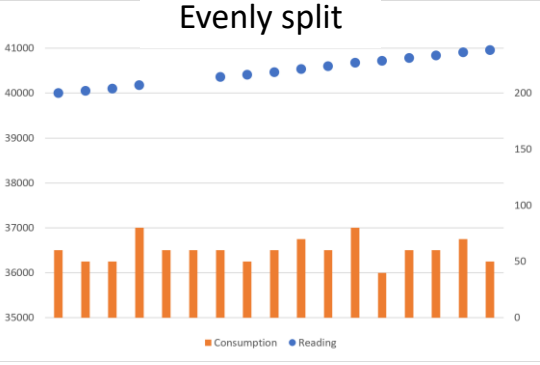
Not filled



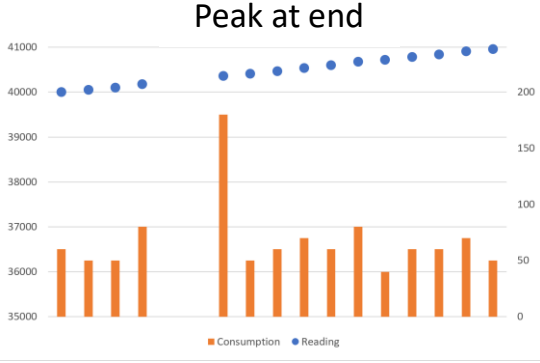
Peak at start



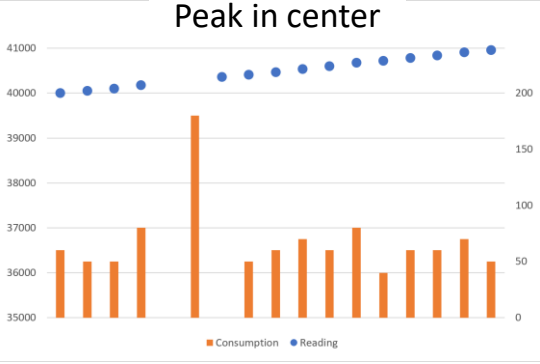
Evenly split



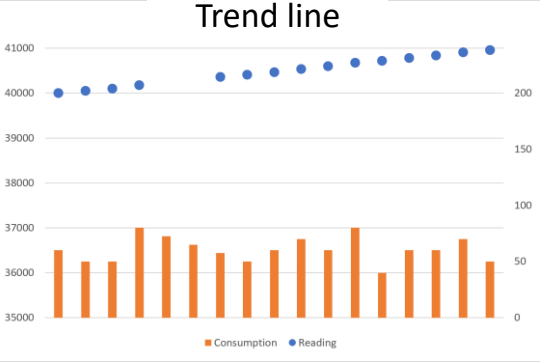
Peak at end

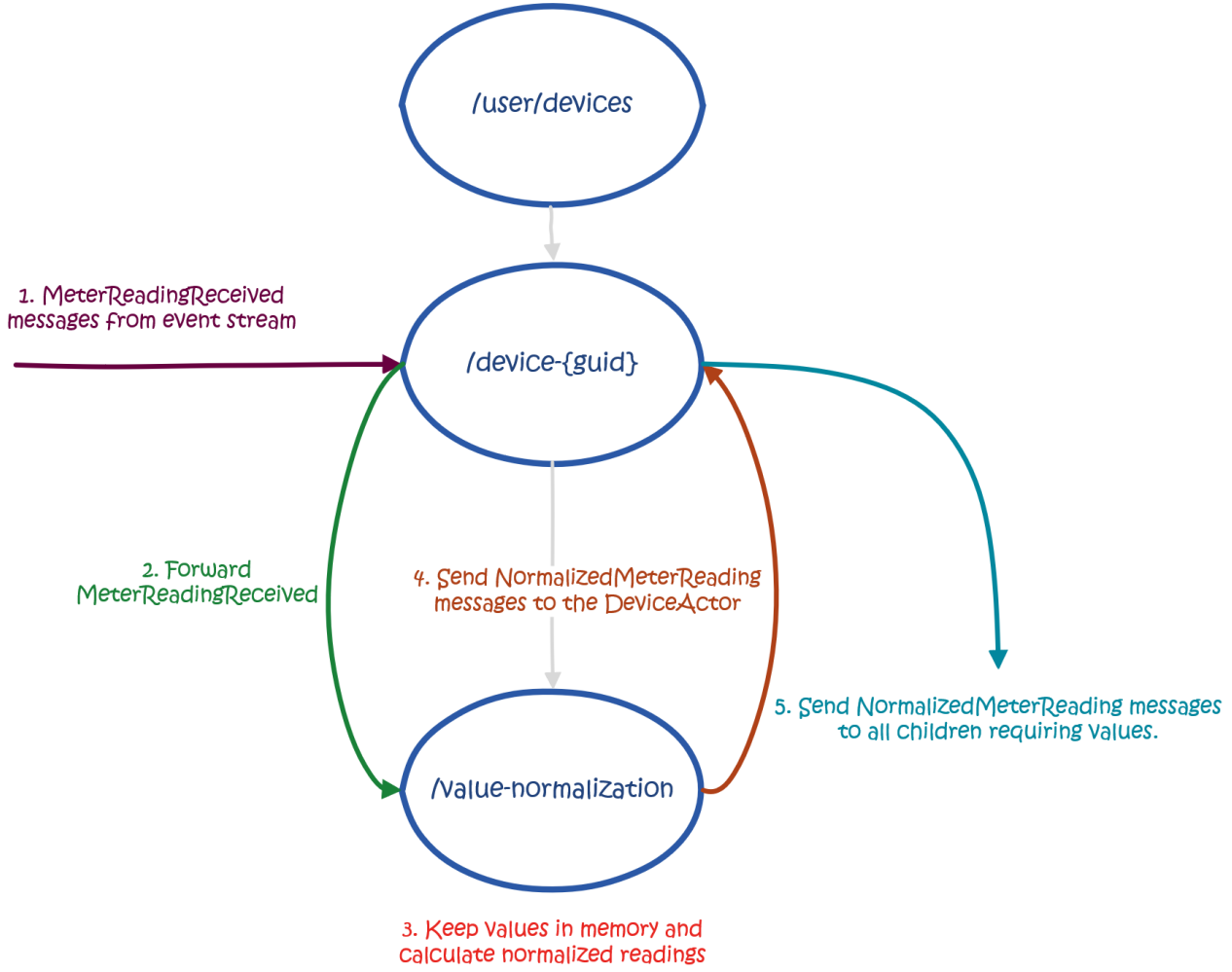


Peak in center



Trend line




/user/devices
1. MeterReadingReceived messages from event stream
/device-{guid}
2. Forward MeterReadingReceived
4. Send NormalizedMeterReading messages to the DeviceActor
5. Send NormalizedMeterReading messages to all children requiring values.
/value-normalization
3. Keep values in memory and calculate normalized readings

AXXES
IT CONSULTANCY

Persisting Data

Saving to disk what cannot be lost

AXXES
IT CONSULTANCY

Akka.Persistence

Actors that recover their state when created:

- Inherit from `PersistentActor`
- Give it a unique `PersistenceId`
- Persist Events with the `Persist(…)` command
- Persist snapshots with the `SaveSnapshot(…)` command
- Register `Recover<T>(…)` handlers to restore state

AXXES IT CONSULTANCY

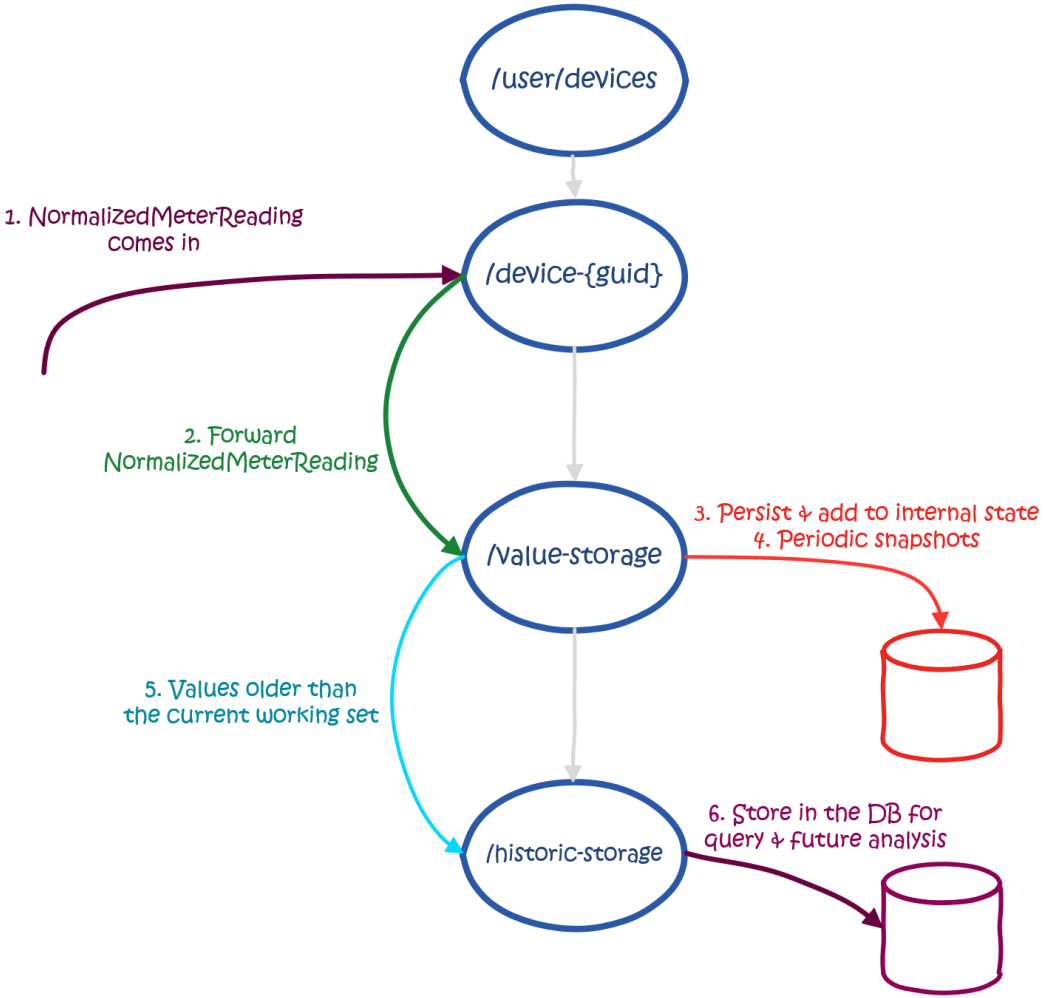
```
public class MyPersistedActor : ReceivePersistentActor
{
    // Any PersistentActor needs a unique key!
    public override string PersistenceId { get; }

    // Grouping state into a state object is a good idea
    private MyState _state = new MyState();

    public MyPersistedActor(Guid id)
    {
        PersistenceId = $"my-persisted-actor-{id}";

        // There's a difference between 'Commands' and 'Recovers'
        Command<MyMessage>(HandleCommand);
        Recover<MyMessage>(HandleMessageInternal);

        // Snapshot events
        Recover<SnapshotOffer>(HandleSnapshotOffer);
        Command<SaveSnapshotSuccess>(HandleSnapshotSuccess);
        Command<SaveSnapshotFailure>(HandleSnapshotFailure);
    }
```

AXXES
IT CONSULTANCY

```
private int _msgSinceLastSnapshot = 0;

private void HandleCommand(MyMessage command)
{
    // Persists the message to the store and the actor simultaneously.
    Persist<MyMessage>(command, HandleMessageInternal);

    // Save a snapshot every 100 messages
    if (_msgSinceLastSnapshot == 100)
    {
        SaveSnapshot(_state);
        _msgSinceLastSnapshot = 0;
    }
}

private void HandleMessageInternal(MyMessage message)
{
    // In recovery, we call this directly, no need to persist it again.
    _state.Add(message);
    _msgSinceLastSnapshot++;
}
```

AXXES
IT CONSULTANCY

```
    private void HandleSnapshotOffer(SnapshotOffer offer)
    {
        if (offer.Snapshot is MyState newState)
            _state = newState;
    }

    private void HandleSnapshotSuccess(SaveSnapshotSuccess success)
    {
        // Handle a successful snapshot save
    }

    private void HandleSnapshotFailure(SaveSnapshotFailure failure)
    {
        // Handle the failure to save a snapshot
    }
}
```


/user/devices
1. NormalizedMeterReading comes in
/device-{guid}
2. Forward NormalizedMeterReading
/value-storage
3. Persist & add to internal state
4. Periodic snapshots
5. Values older than the current working set
/historic-storage
6. Store in the DB for query & future analysis

AXXES
IT CONSULTANCY

Restart Behavior

How can we be sure to get going again after a restart?

AXXES
IT CONSULTANCY

After a system restart

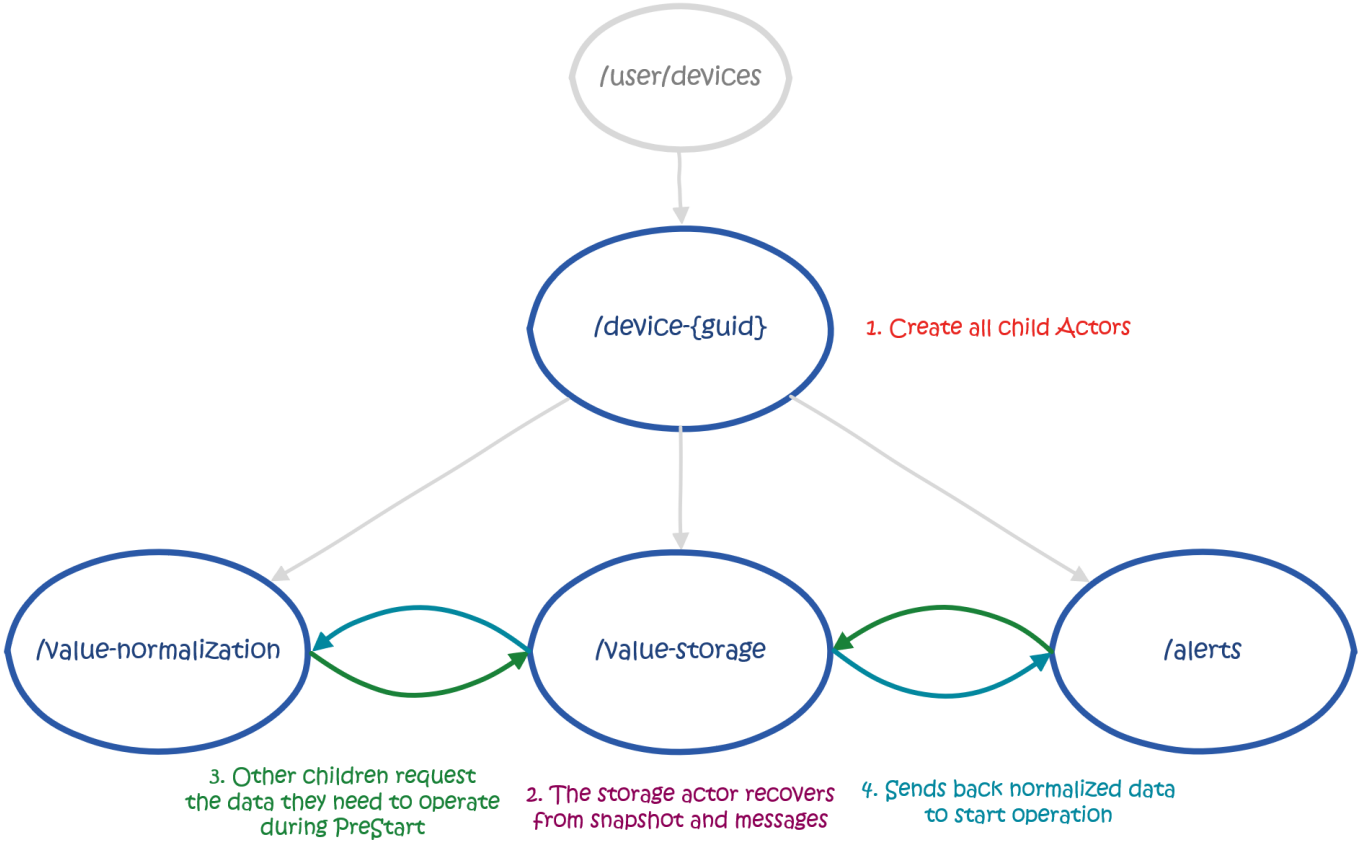
Recreating Actors isn't that hard:

- Query the DB on startup
- Create the required Actors

How do we get all our Actor state back?

- Minimize the amount of actors that need to recover state
- 1 PersistedActor per device = ideal
- Let other actors query that actor for the state they need

AXXES
IT CONSULTANCY



Beyond this talk …

The stuff that we didn’t talk about …

IT IS ABOUT PEOPLE

AXXES
IT CONSULTANCY

Make Akka.NET production ready

- **Configuration:**
  HOCON
- **Clustering:**
  Run across multiple machines
- **Logging:**
  Adapters for Nlog, SeriLog, etc.
- **Dependency Injection:**
  Akka.NET supports DI for your actors
- **Production monitoring:**
  Phobos

AXXES
IT CONSULTANCY

Start learning

- **FREE Akka.NET Bootcamp by Petabridge:** https://github.com/petabridge/akka-bootcamp
- **PluralSight courses:** There are some good courses on there!
- **Petabridge blog:** https://petabridge.com/blog/
- **Petabridge remote training (paid):** Worth it when you have serious questions

AXXES
IT CONSULTANCY

Deployment

1. Pause the process that reads from the event stream
2. Wait for processing to end
3. Deploy the Akka.NET cluster
4. Re-create actors (triggering Persistence restores)
5. Resume sending from the event stream

→ When done right, you can do this without losing data!

AXXES
IT CONSULTANCY

Conclusion

1. Check if your problem domain is a fit for Actors
2. Decide which part of the solution will be Akka.NET
3. Design your actor hierarchies appropriately
4. Normalizing data helps a lot
5. Think about deployment & recycles

AXXES
IT CONSULTANCY

Ask me some questions!

AXXES
IT CONSULTANCY

About

Hannes Lowette

.NET Consultant & Competence Coach at **@Axxes_IT**

@hannes_lowette

#20086521

Code samples and slides at :

github.com/Belenar/Axxes.AkkaDotNet.SensorData

AXXES
IT CONSULTANCY

Pictures from:

https://thetravelquandary.com/
https://imgflip.com/
https://commons.wikimedia.org/
https://akka.io/
https://codingjourneyman.com/
https://getakka.net/
https://dotnet.github.io/orleans/
https://blogs.msdn.microsoft.com/
https://www.karlrupp.net/
https://www.xkcd.com/
http://www.last.fm/
https://www.newscientist.com/
https://io9.gizmodo.com/
https://www.thebestcolleges.org/