

# But, Does It Scale?

Clustering, Load Balancing, Tuning and  
Performance Considerations

- Architect for Intalio
- Specialized in the server runtime
- Apache Member
- Chair for Apache ODE (Orchestration Director Engine)

- What type of performance do you need?
- Design tips
- Intalio Runtime components detailed
- In-memory and persistent processes
- Footprint of a process
- Server Clustering

# What's Performance Anyway?

- Scalable?  
A few more machines never hurt
- Fast?  
This has to be done in no time
- Lightweight?  
Do you fit in my iPhone?

- Throughput vs. Latency
- One more user, no big deal  
vs.
- I need my reply fast
- You can't optimize for both

- Measuring by transaction
- An atomic unit of work
  
- The questions become:
  - How many transaction per second/minute/hour/day/week/month?
  - How long the average transaction takes?

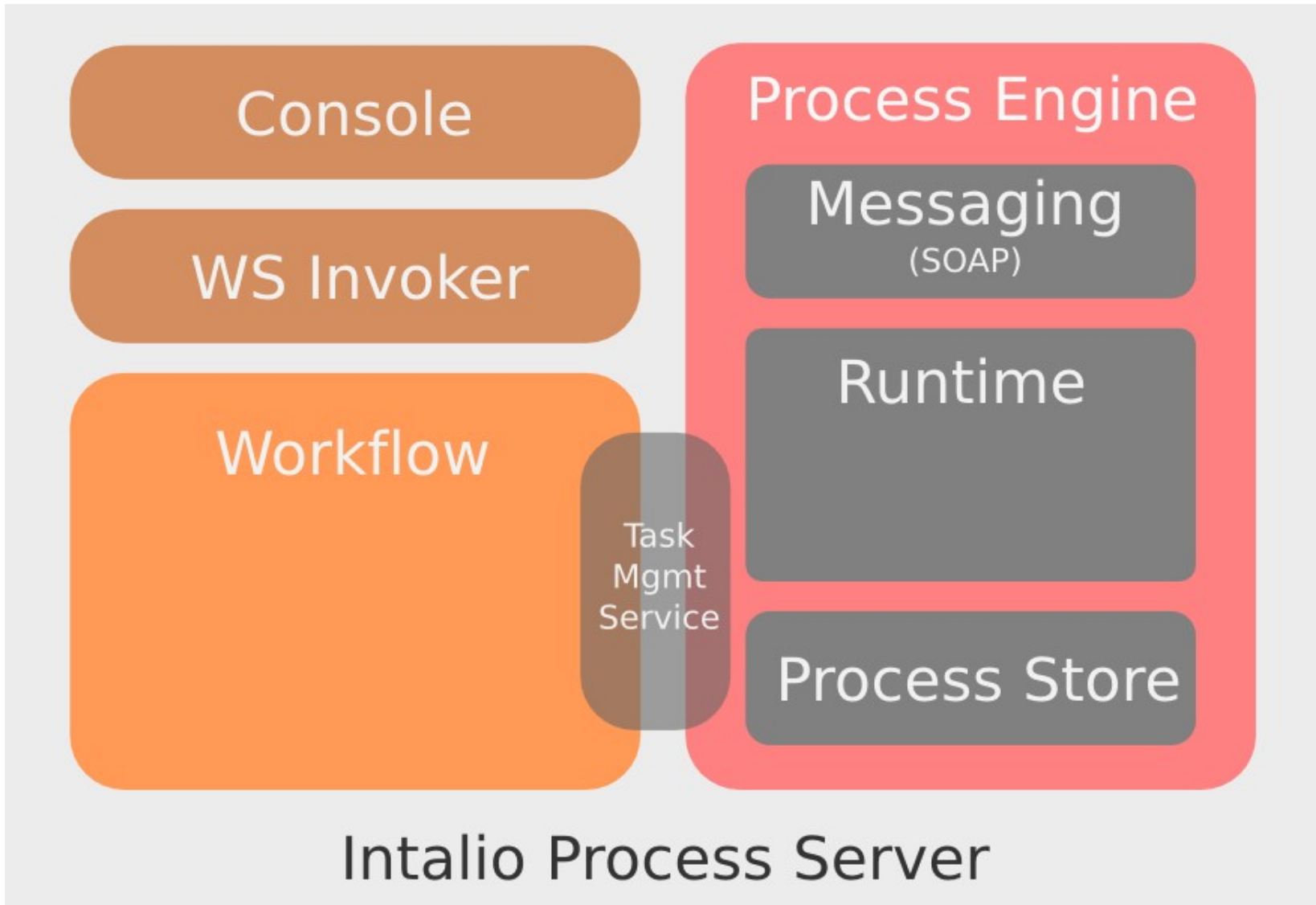
- Where are the bottlenecks?
- A bottleneck defines the limits of the whole system
- I/O (disk, network) or CPU bound
- Don't sweat it: it's the database!

# Design Tips

- Short-lived processes: in-memory (stick with request/response)
- Lots of assignments get expensive, XSL when you can
- Longer processes: adopt one-way asynchronous calls
- Parallelize when possible (less transactions)
- Try to limit the amount of data (with XML, I know it's hard)

# Intalio Server Components

Who is slowing down everybody?



- Workflow: humans are slow
- Process Store: only matters for clustering
- Messaging layer
  - Important for latency
  - Doesn't matter much for throughput

- What? Messaging doesn't matter?!
- General case: processes are fast enough, you just want a lot of them
- Shared memory space or thousands of miles away: same throughput
  
- POJO, JBI, SCA, Web Services? Performance not an issue.
- YMMV of course

- The culprit: the engine
- Why? The database of course!
- What does the engine do?
  - Route and save messages
  - Load instances
  - Manipulate and save data
  - Save instances

- And what's the process data?
  - Messages and variables
  - Process instance related: scopes, correlation, ...
  - Instance state
  
- Instance state and instance data have different lifecycles

# In-Memory and Persistent Processes

- The bottleneck is in the DB: remove the DB
- All runtime data transient in memory (messages, variables, instance, ...)
- Configurable on a per-process basis
- Numbers
  - Persistent process: 3.6M tx/day
  - In-memory: 14.3M tx/day
- But...

## ■ Limitations

- Only for request response interactions
- Instances can't be queried or retrieved
- Long interactions not advised

■ Keeping instances around = one giant memory leak

- Back to persistence: how can we make it better?
  - Persisting less: tuning events generation
  - Tuning your database: talk to your DBAs
  - External Variables: divide and conquer

# Footprint of a Process

- A process in memory
  - The process skeleton, just what's needed to receive messages
  - The static definition a.k.a. Omodel
  - Activities and data only during execution
- Most of the memory consumption comes from the process definition

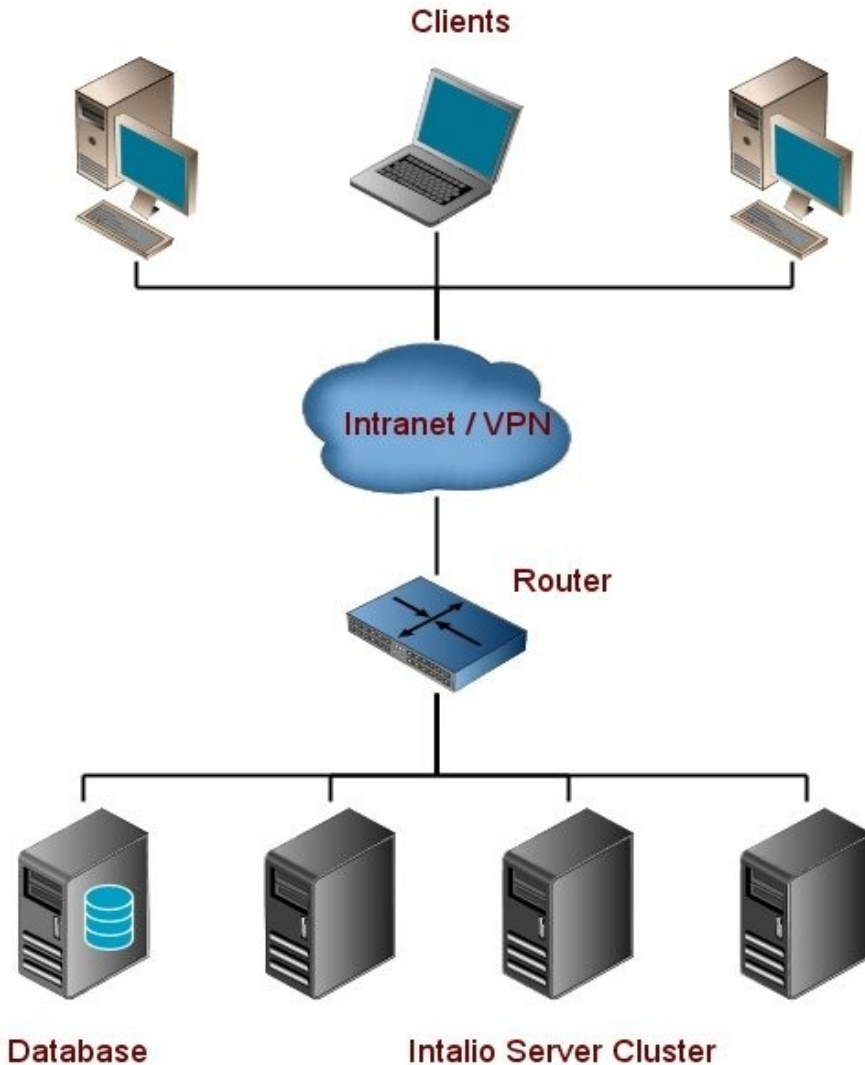
- Lazy-loading of processes
  - At startup, only the skeletons get created
  - Definitions are only loaded when needed (first message).
  
- Advantages
  - Faster startup
  - Processes that aren't used consume almost no memory

- Processes dehydration
  - Definitions are discarded from memory when a process isn't used for a given (configurable) time
  - When the process gets used again, the definition is reloaded
  
- Advantage
  - Much lower memory footprint at any give time
  
- Dehydration and lazy-loading add a small overhead when the definition is loaded

# Clustering

- Two different ways of scaling:
  - A few big giant machines
  - Many cheap machines
- Google adds 500,000 servers a year, maximizing CPU Power / Cost
- Second strategy offers bigger bang for the bucks

- Failover: if one machine fails, others can help
  - Active/passive configuration: only one server runs, requests get routed to a backup on failure
  - Active/active: all nodes run in parallel, requests get routed to others on failure
- Load-balancing
  - Sharing some of the I/O and CPU load



- Clients connect to the server through a load balancer
- Servers automatically detect each other by multi-casting
- All servers share a database

# Screencast

<http://bpms.intalio.com/files/resources/ClusteringDemo/ClusteringDemo.htm>

- Scale linearly: limit synchronization between nodes
- Affinity: a message aimed at a given process is always routed to the same server
  - Delicate for BPEL: correlation
- Sharding: separate data in subsets (for each process), databases love it!
- Not implemented yet!

# Questions?