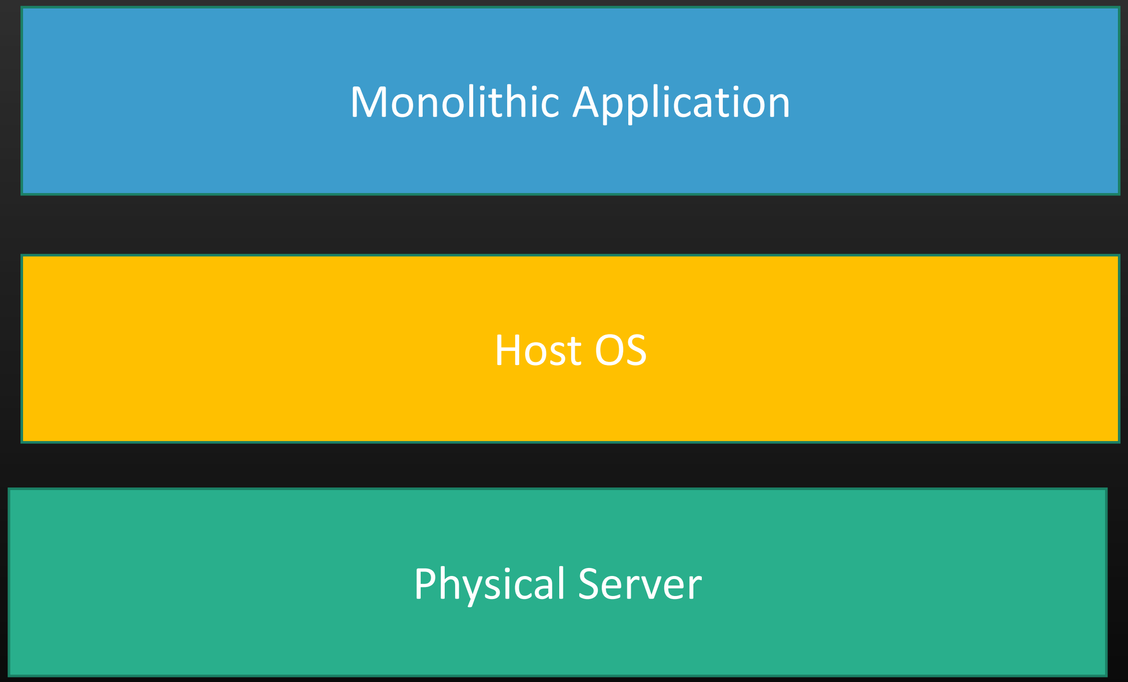
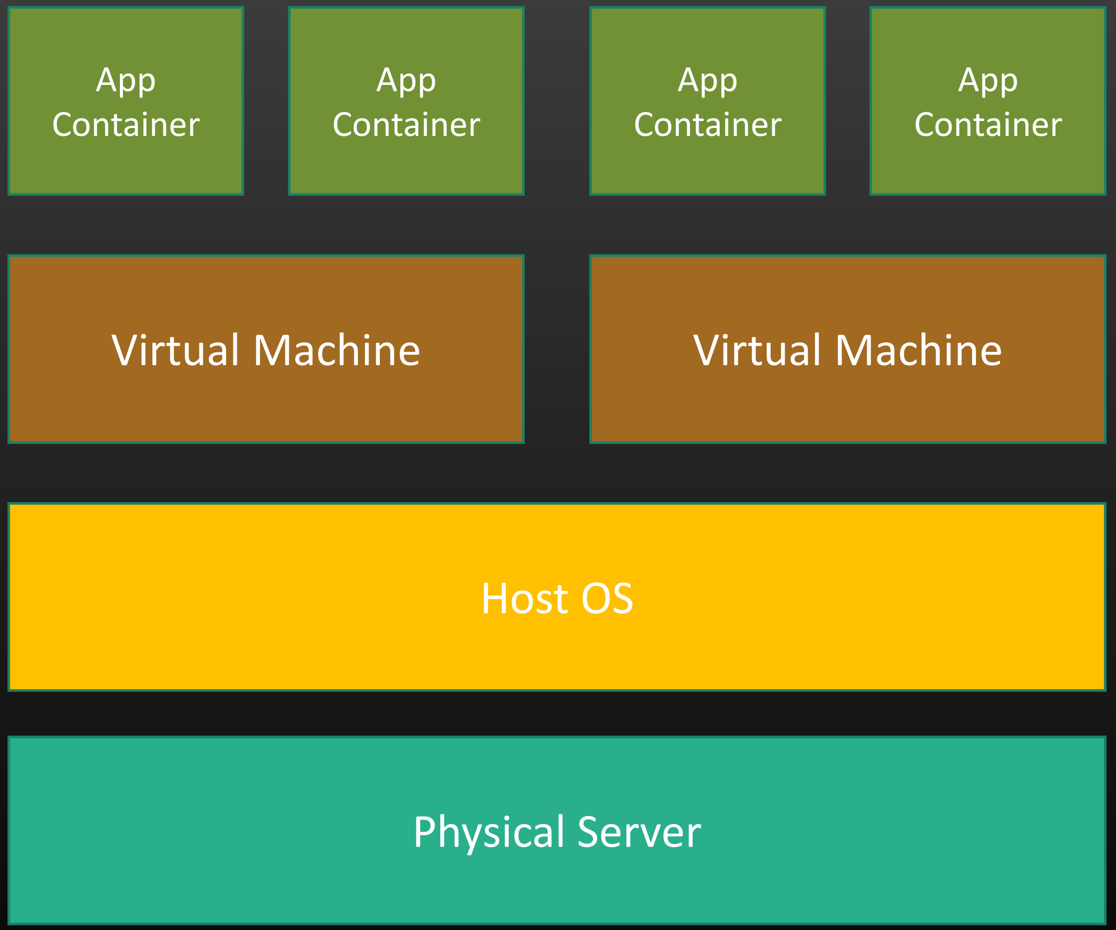
How .NET Runtime Evolves for the Cloud

Mei-Chin Tsai

Workload such as Exchange, Bing



Workload such as Lambda or Functions



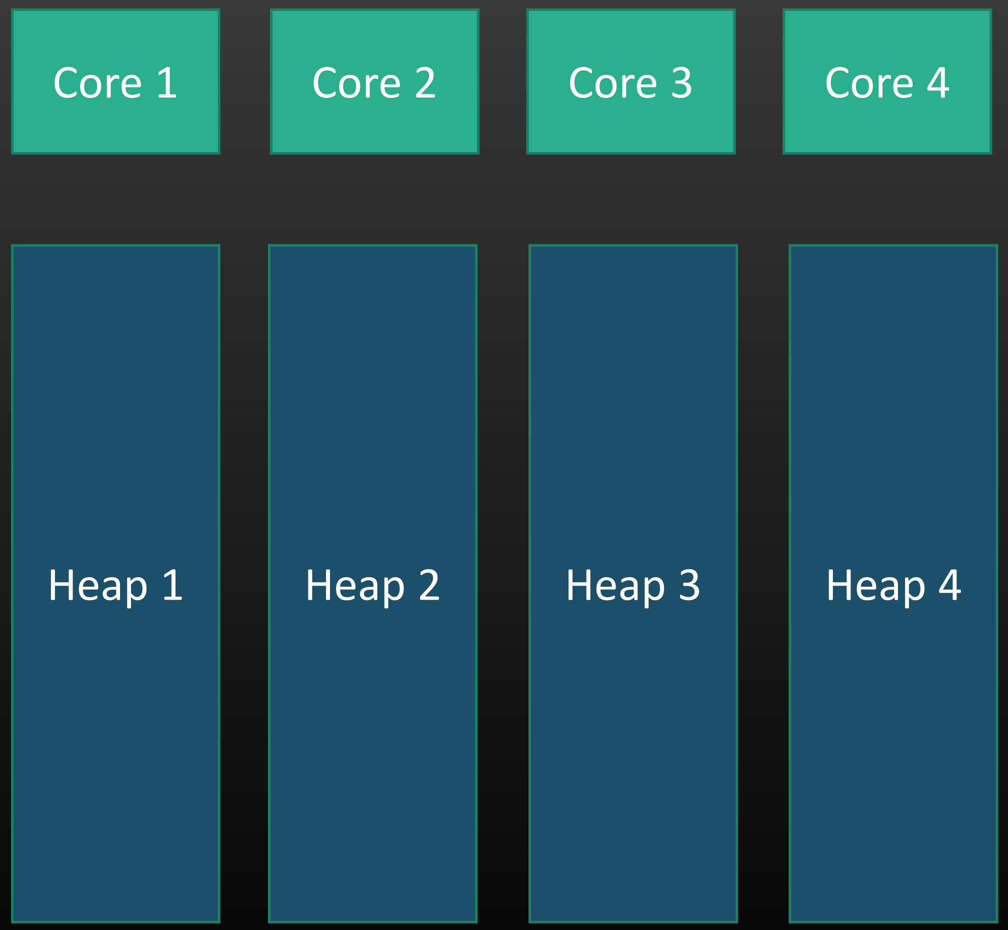
Physical resources that impact Runtime heuristics

- Number of available CPU cores
  - Number of threads
  - Number of managed heaps
- Size of available memory
  - Heap size
  - Number of heaps
- Others

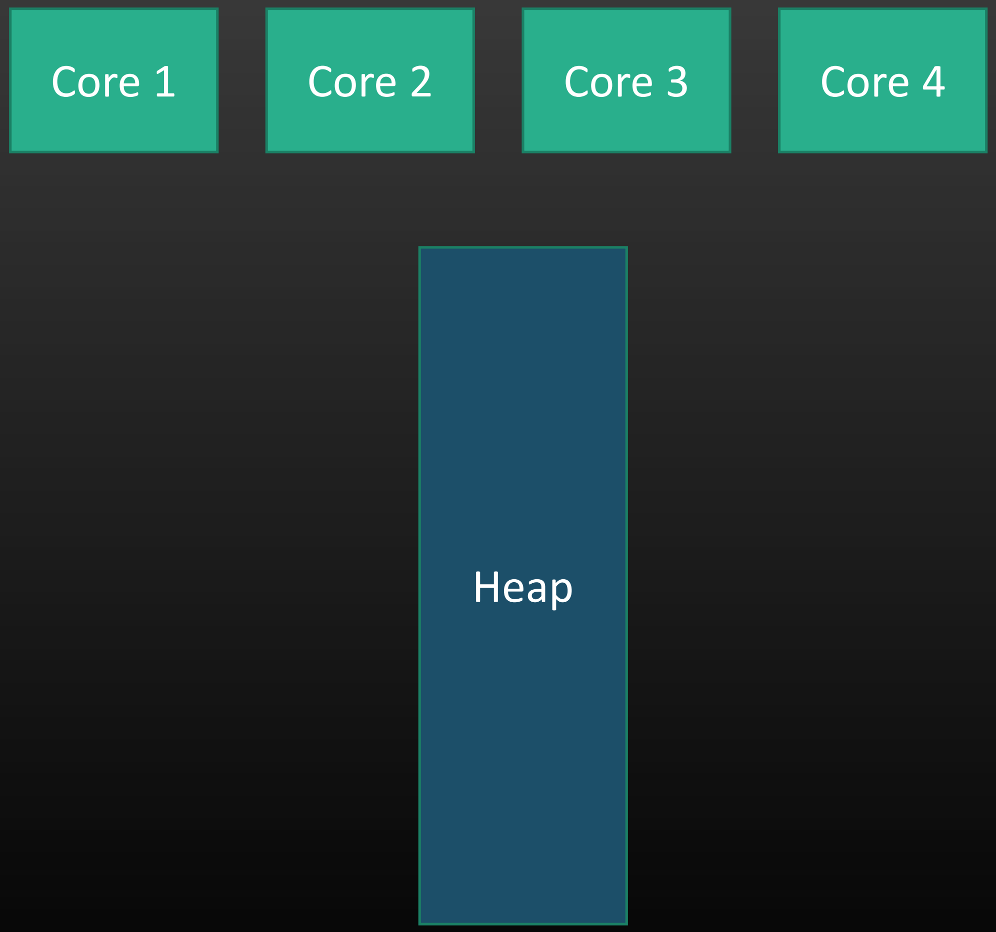
.NET GCs

- .NET GCs are generational
- Two different flavors of GCs today
  - Workstation GC
    - One managed heap (one GC thread)
  - Server GC
    - N managed heaps and N GC threads

Server GC
one GC heap per core



Workstation GC
one heap for all



Use multi-pronged approach for scaling

Runtime

- Using less memory is generally better

Application/Process Configuration

- Scale down
  - Docker support

Application

- Allow application to specify intent

Runtime

- Scale up
  - Optimize for many-core chip architecture

Using less memory is generally better – less memory by default

- Reduce the initial commit size of gen 0
- Reduce the initial gen 0 allocation budget to better align with modern cache size and cache hierarchy
- New policy to determine number of GC heaps to create based on memory limit
  - Example –
    - Application memory limit is 160MB, default GC memory segment per heap is 16MB
    - Old behavior: allocating one heap per core on 48 core machine exceeds limit
    - New behavior: allocate 10 heaps, meets limit

TechEmpower benchmarks
~50% of committed memory reduction

	Plaintext	Json	Plaintext N-P	MVC Plaintext	MVC Json
Memory	3.0 vs 2.2 601 MB 1,336 MB (-55 %)	3.0 vs 2.2 526 MB 1,187 MB (-55.7 %)	3.0 vs 2.2 515 MB 1,181 MB (-56.4 %)	3.0 vs 2.2 678 MB 1,274 MB (-46.8 %)	3.0 vs 2.2 546 MB 1,211 MB (-54.9 %)

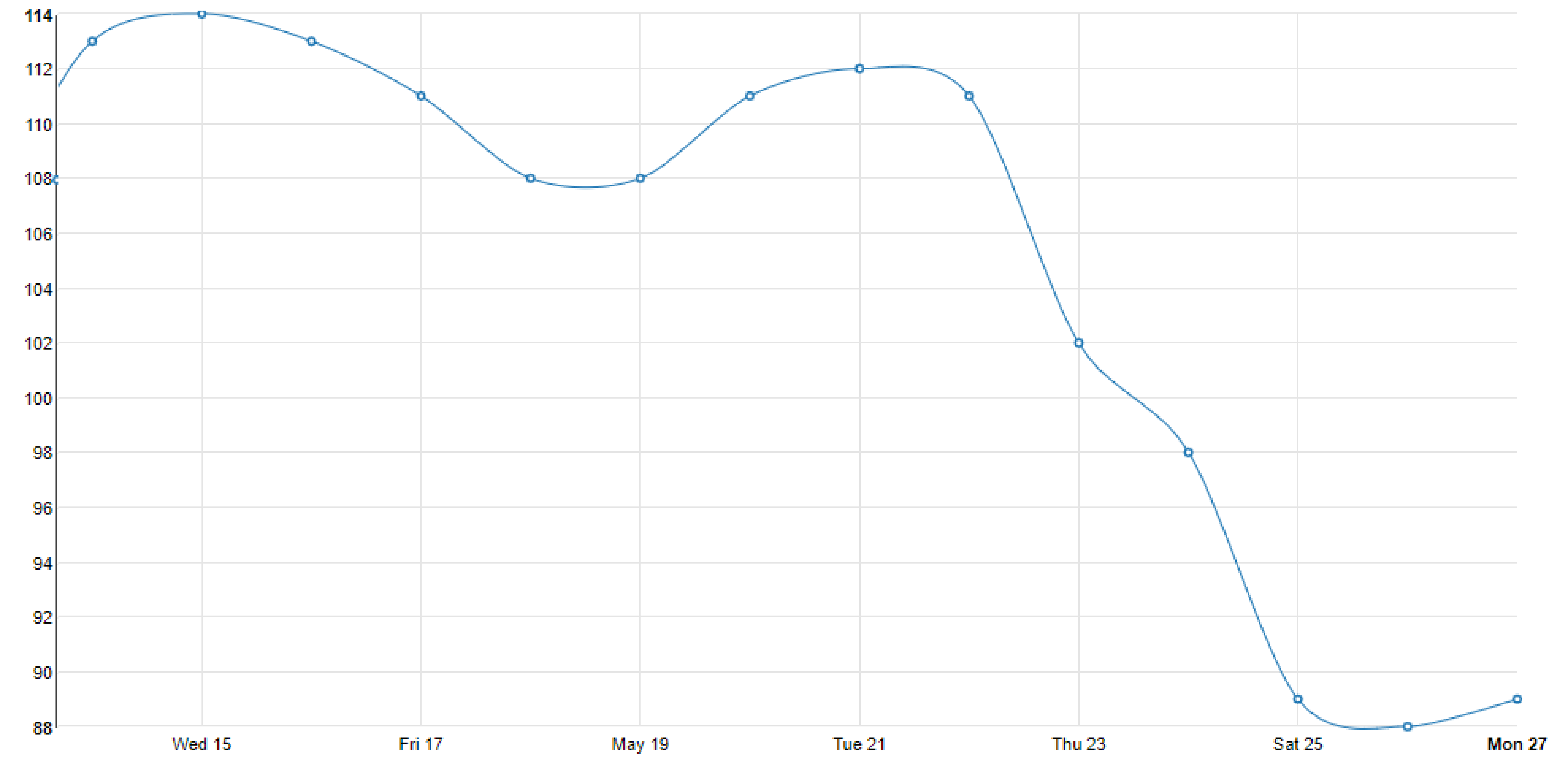
Scale down – Docker container support

- Memory limit set on container
  - docker run -m 100mb -t xxx
- GC heap is not the only component use memory.
- Introducing GCHeapHardLimit config
  - GCHeapHardLimit - specifies a hard limit for the GC heap
  - GCHeapHardLimitPercent - specifies a percentage of the physical memory this process is allowed to use
- If neither is specified but the process is running inside a container with a memory limit specified, we will take this as the hard limit:
  - max (20mb, 75% of the memory limit on the container)

Allow application to specify intent - Large pages support

- Observation - Bing frontend observed many TLB misses in their workload latency
- Add an application config to allow large page support
- Pay more cost on each new page load request but hope to pay less frequently
- On Windows – Runtime commit all the managed memory upfront.
- Does change application performance characteristic
  - Use carefully

Bing frontend (SNR) –
P95 improvement ~108ms -> ~88ms (18.5% improvement).
50th %ile (average), the improvement was around 9%



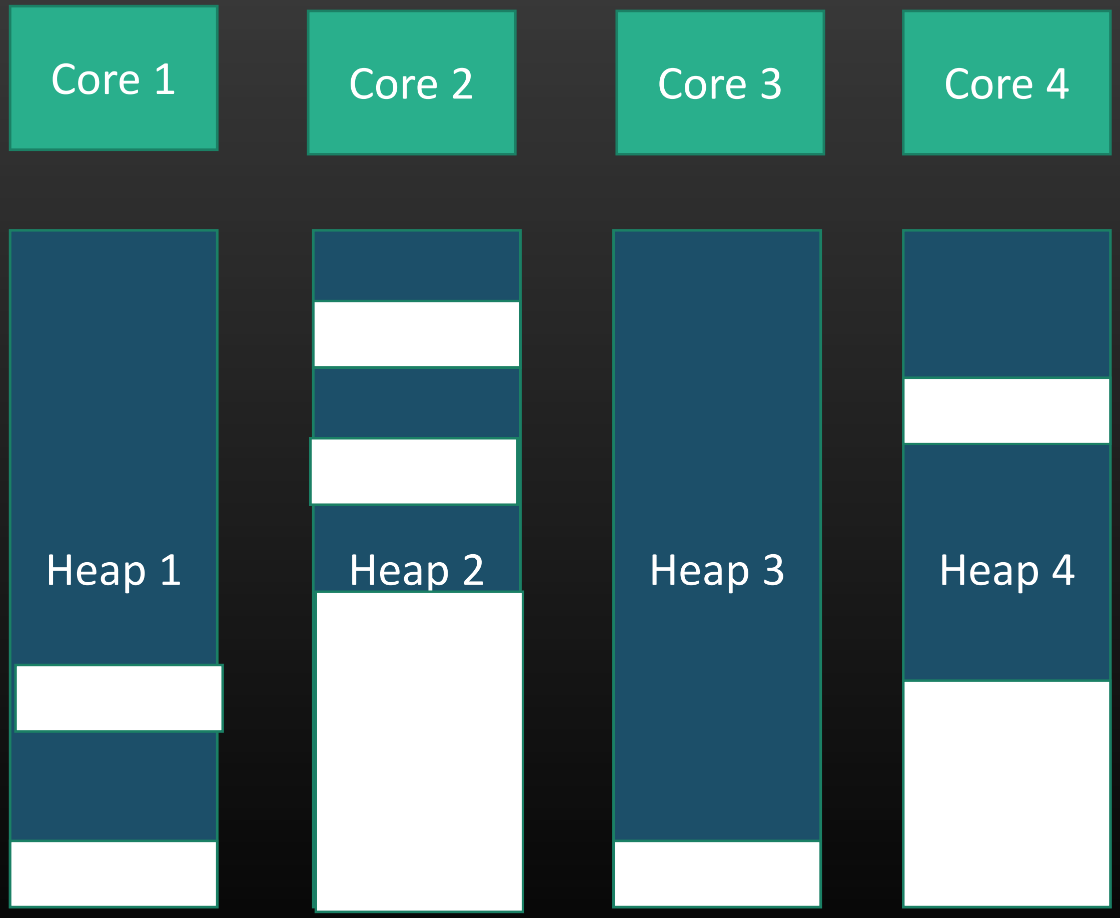
Scale Up – many-core processors

Trend is to use more cores (many of our customers are on 32 to 48 cores and are looking to upgrade core count)

E.g. AMD ROME CPU – 64 cores, NUMA

The heap balancing mechanism needed to be revisited

Server GC
one GC heap per core



Each heap maintains its gen0 budget (ie, allocations it allows before triggering the next GC)

- when any heap's budget is exceeded, a GC pass is triggered
- When GC is triggered, the whole world is stopped

Heap balancing goal

- When allocations on threads are balanced, they should stay allocating on the same heap
- When allocations on threads are unbalanced, they should in general spread evenly across heaps
  - But there are special considerations, eg, we should favor the heap for that core

Current heap balancing mechanism explained

- Home and alloc heap
- Local heaps (on current NUMA node) vs remote heaps
  - Look at local heaps first
  - Requires a large delta to balance to a remote heap
- When allocating to a remote heap, we incur not just remote allocation cost. We also incur remote access cost in the future.
- Problem – we are trying too hard to keep heaps well balanced
  - Not showing up as problems when you had fewer heaps to search
  - The cost of remote access cannot be easily factored in ahead of time

Realizations

- If we do less work and still achieve similar fill ratios, we should do that instead of looking at each heap
- Balancing on earlier allocations is less important than later ones which tend to survive more

Thoughts

- Really need better tooling to help with the investigation
  - vtune does show many memory counters but they can be hard to interpret; we also want to correlate with GC activities
  - New GC specific tooling shows how threads and their alloc heaps migrate

Show the heap/thread logs of runtime instrumentation

Q/A