accu
2023

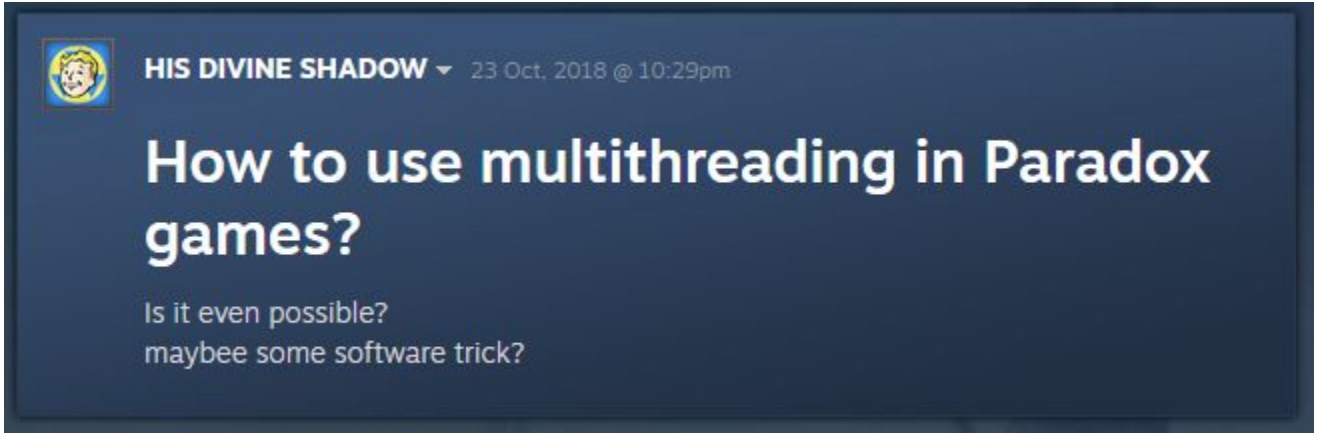
MULTI THREADING MODEL IN PARADOX GAMES:

PAST, PRESENT AND FUTURE

MATHIEU ROPERT

Multi Threading Model in Paradox Games

Past, Present and Future



HIS DIVINE SHADOW 23 Oct, 2018 @ 10:29pm

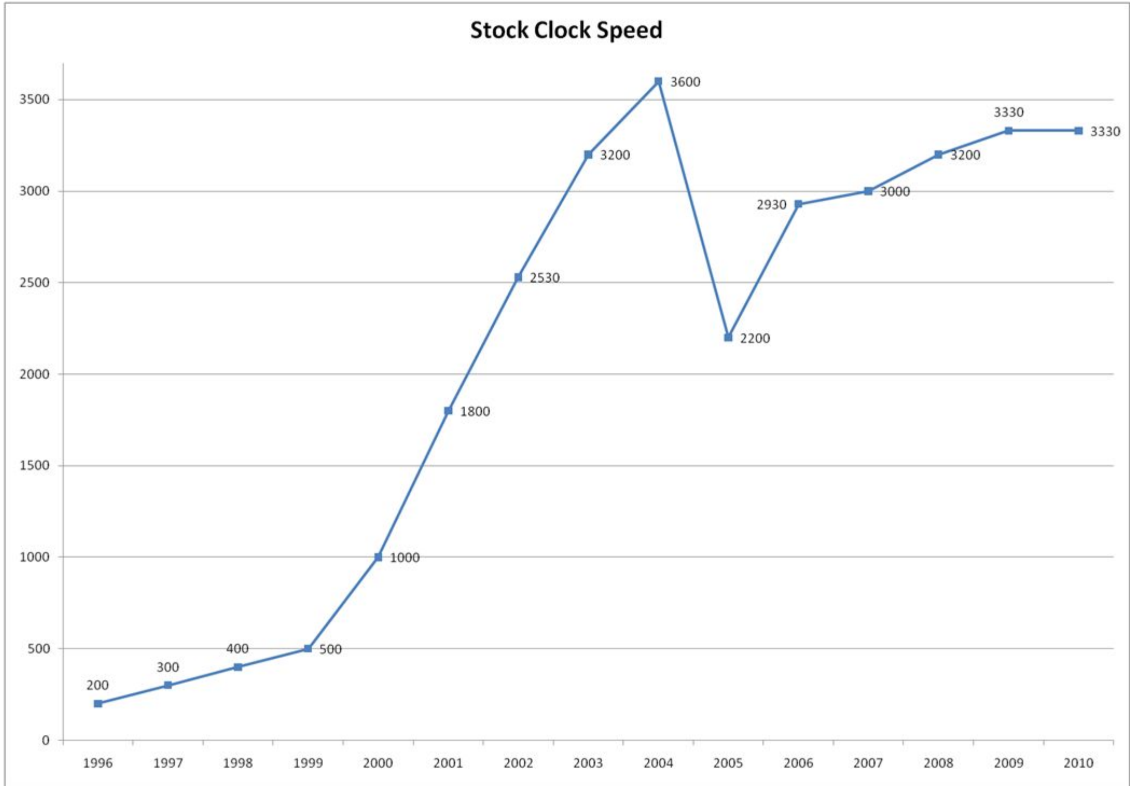
How to use multithreading in Paradox games?

Is it even possible?
maybee some software trick?

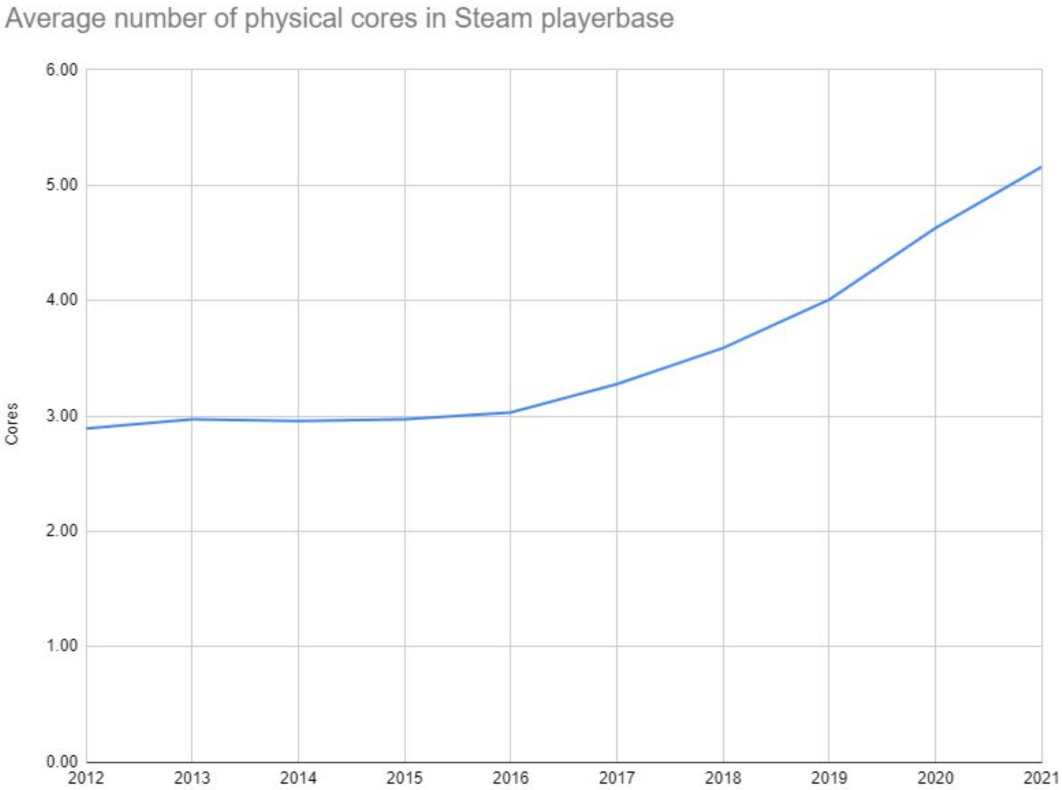
How indeed?

The greatest trick

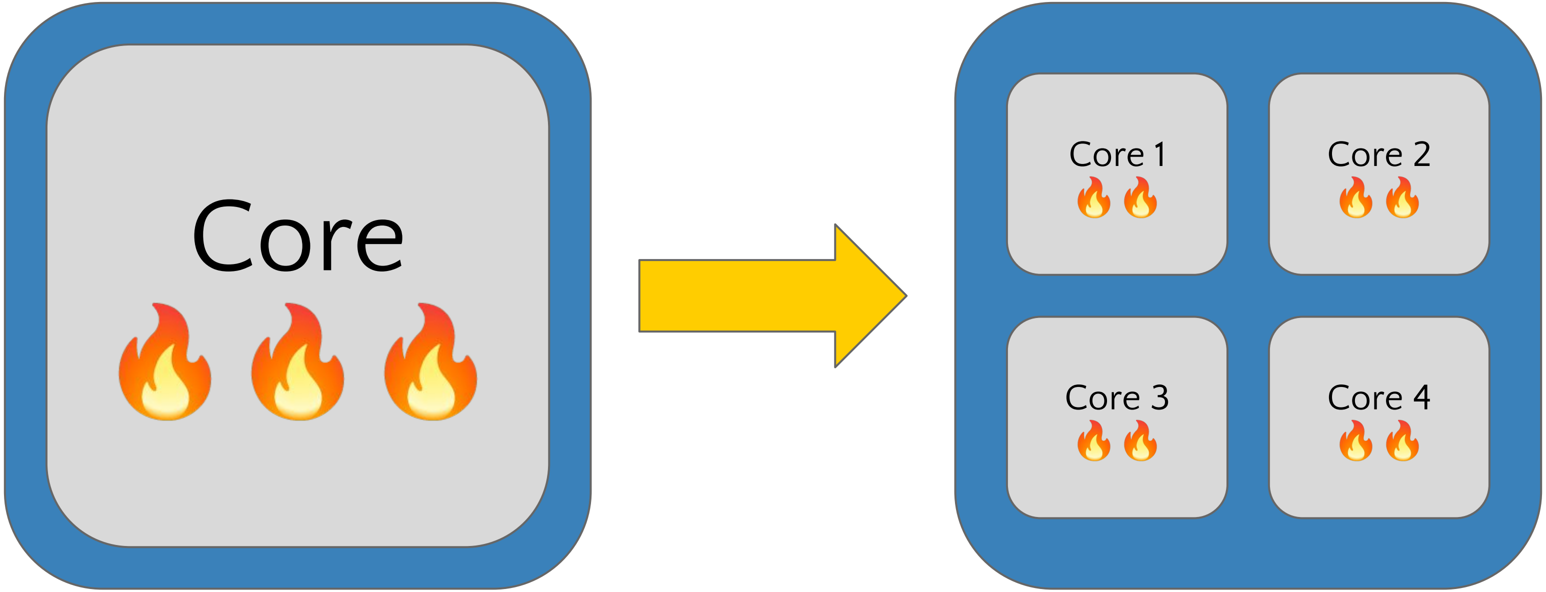
- At level 7 pick Improved Software Trick
- At level 13 pick Greater Software Trick
- Consider the Software Trickster prestige class
- Best done with Gnomes or Catfolks


Stock Clock Speed
3500
3000
2500
2000
1500
1000
500
0
200
300
400
500
1000
1800
2530
3200
3600
2200
2930
3000
3200
3330
3330
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010

Stagnation?



Another direction



Core

🔥🔥🔥

Core 1

🔥🔥

Core 2

🔥🔥

Core 3

🔥🔥

Core 4

🔥🔥

New hardware, new challenges

Hello!

I am ***Mathieu Ropert***

I'm a Tech Lead at Paradox Development Studio where I make Hearts of Iron IV.

You can reach me at:

- ✉ mro@puchiko.net
- 🐦 @MatRopert
- 🌐 https://mropert.github.io

About this talk

- The importance of multi-threading
- Concurrency models…
- … in practice
- Tips & tricks

Profile Machines

Dev Workstation

- Intel i7-7700
- 4 cores / 8 threads
- nVidia GTX 1060
- Optick profiles

Home PC

- Intel i7-10700
- 8 cores / 16 threads
- nVidia RTX 2080 Super
- vTune profiles & Demos

Presentation Machines

Conference Laptop

- Intel i5-4300U
- 2 cores / 4 threads
- Intel HD Graphics
- Google Slides

Remote Workstation

- Intel i7-12700
- 12 cores / 20 threads
- nVidia RTX 3060
- Demos (hopefully)

1

Why multithreading?

More than one use case

The threads you know

- Concept is quite old
- `pthreads` were introduced in 1995
- Most (all?) software engineering classes will cover the basics

The threads you forgot

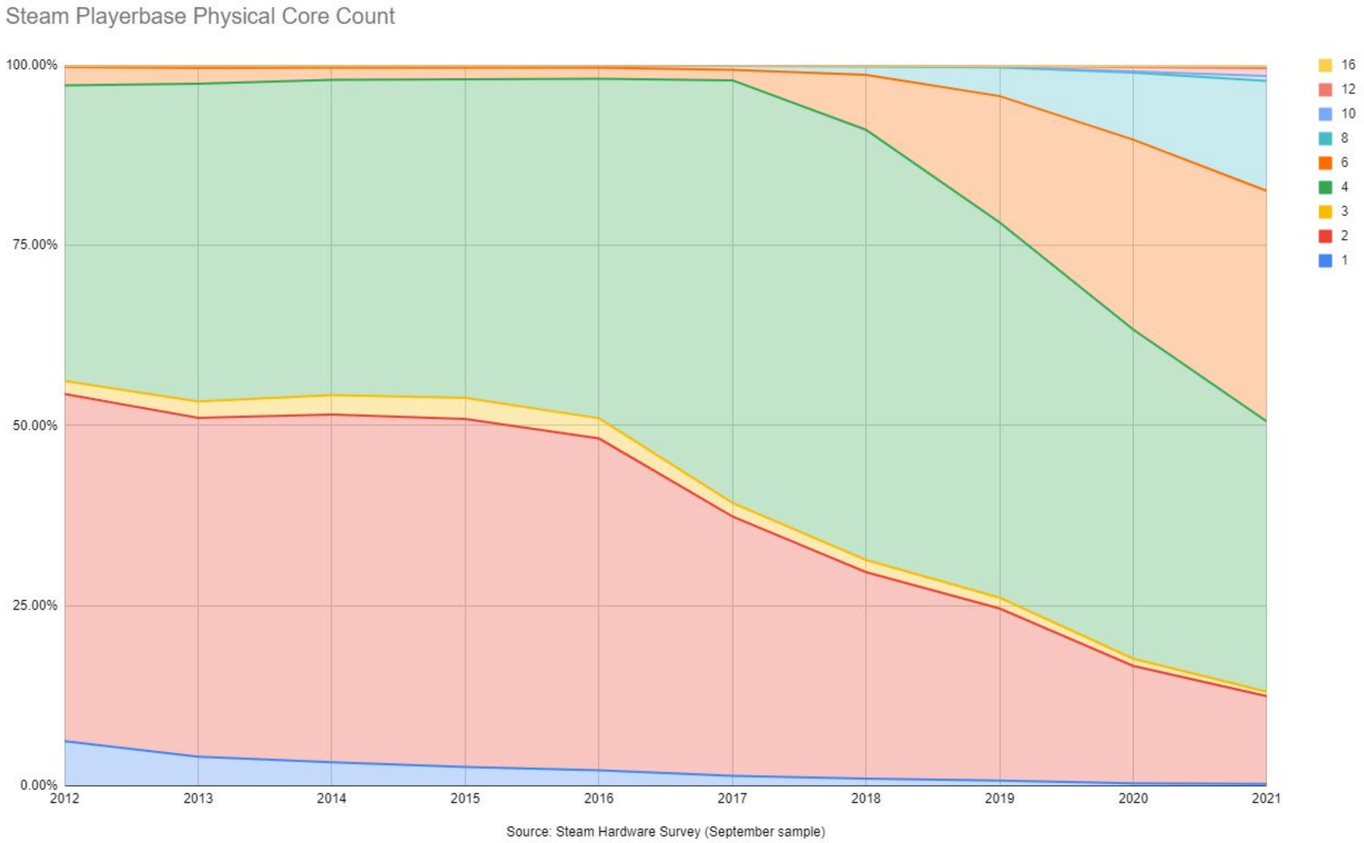
- Desktop machines with more than one CPU only appeared late 2002
- Consumer CPUs with more than one core came up in 2005
- 2+ cores became default for Intel only in 2010

Threads limitations

- Efficiency is limited by the number of CPUs
- Over-subscription worsens performance
- Don’t spawn more work threads than you have cores

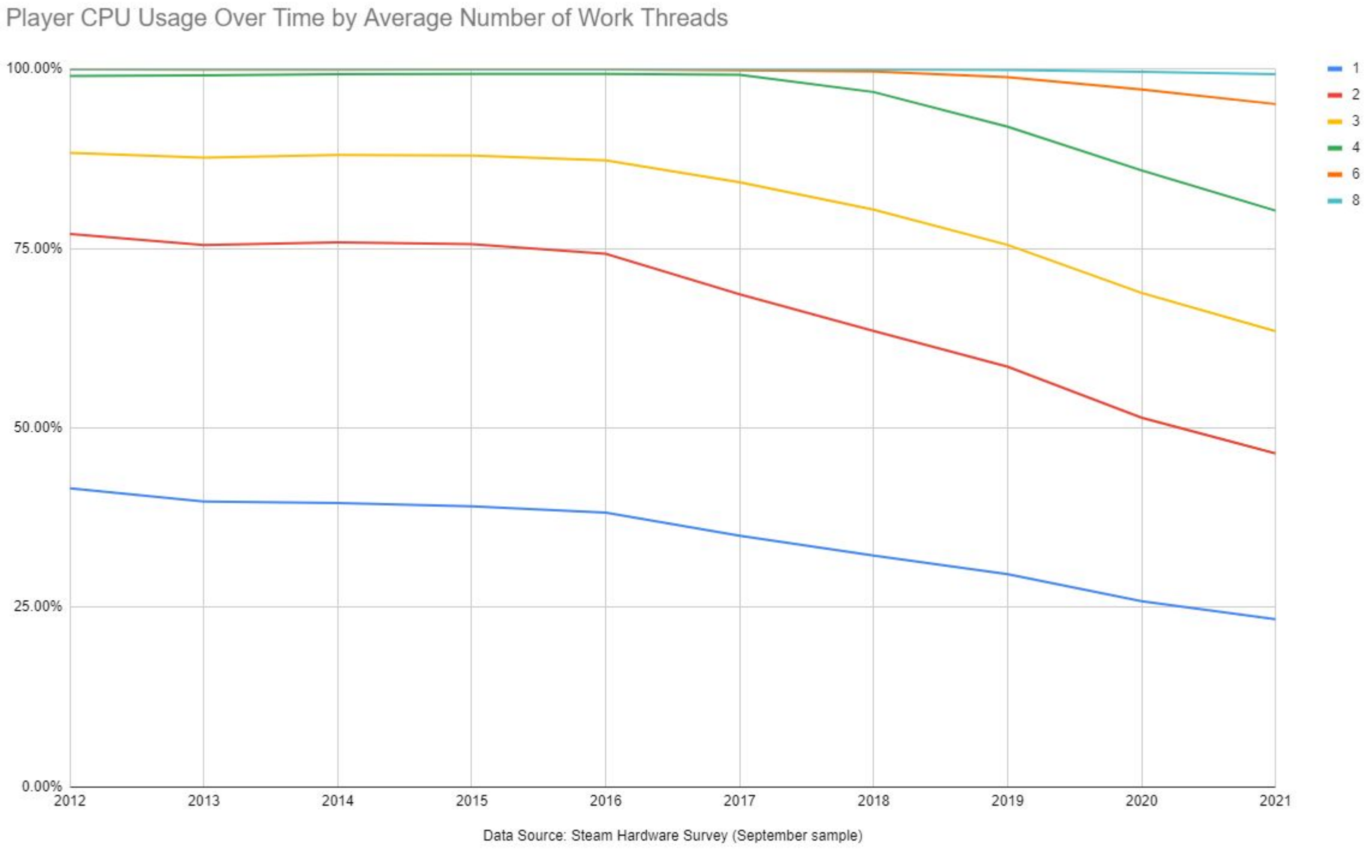
Desktop multithreading

- For a long time the average desktop only had 1 or maybe 2 CPUs
- Good multithreaded code is harder to write
- Use threads for async operations, keep most of the busy work on the main thread



Source: Steam Hardware Survey (September sample)

Times are changing



Times are changing

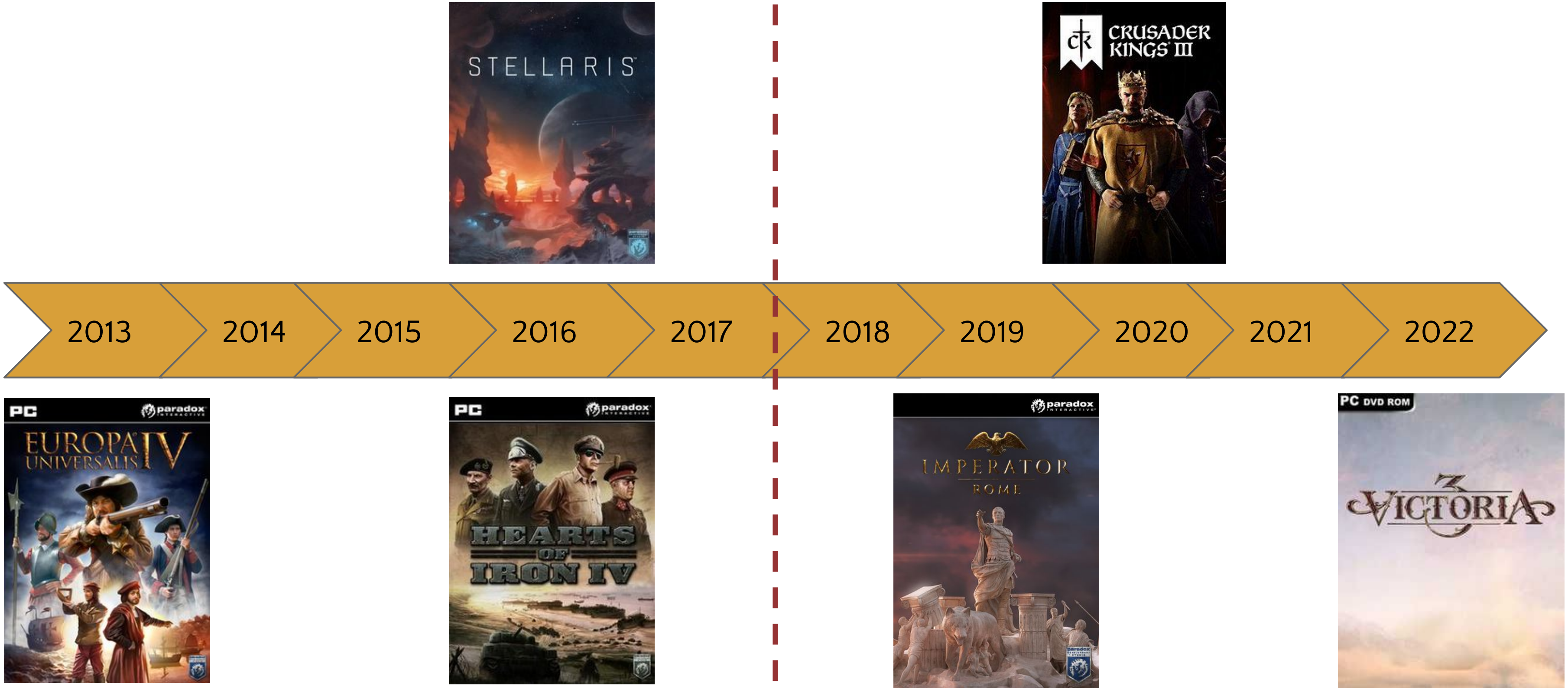
Multithreading today

- Mono thread computation only utilizes about 25% of the userbase processing power
- Multithreading computation is not just a bonus for high-end desktops anymore
- Code needs to adapt

2

Paradox Games and Multithreading

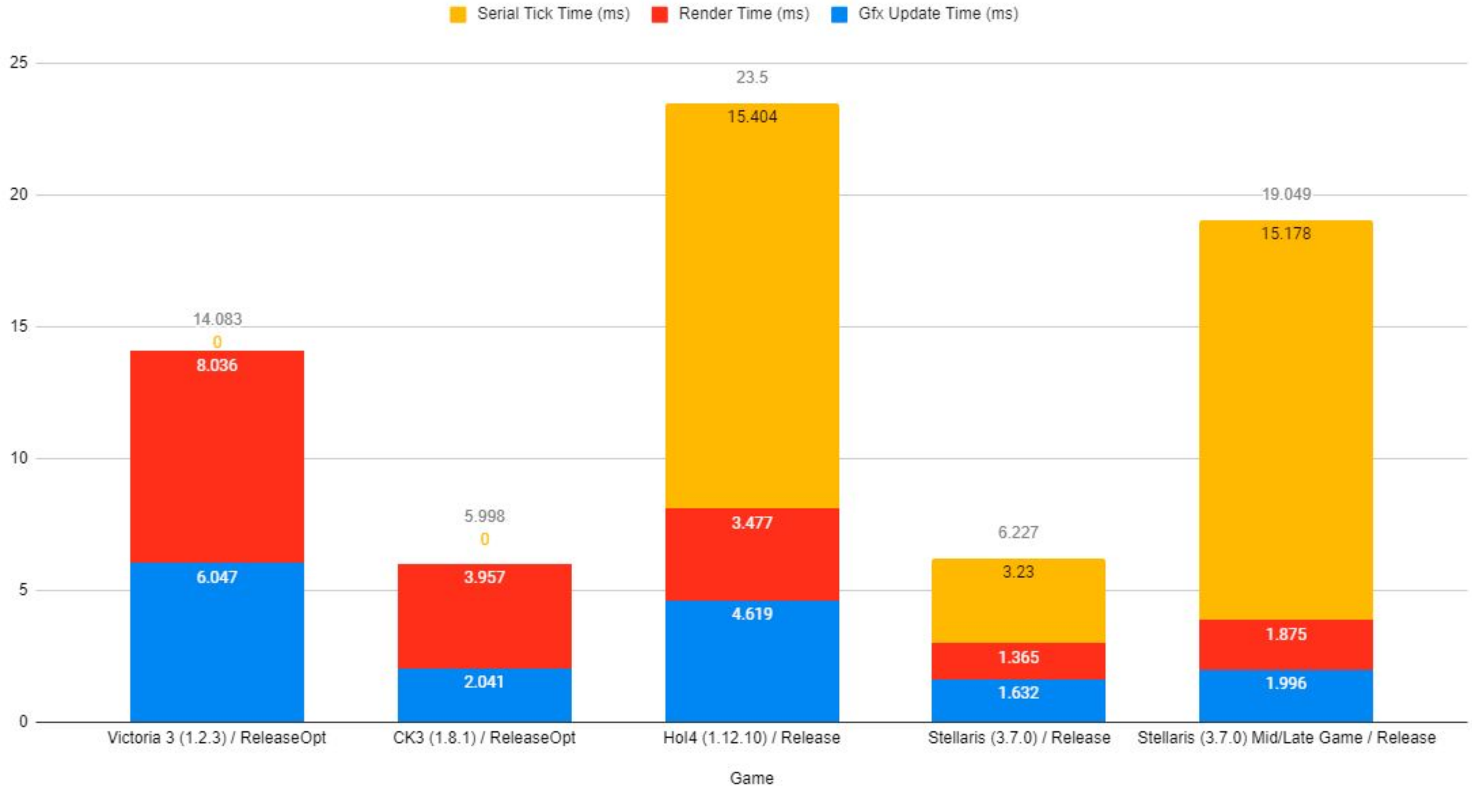
A history of historical strategy games

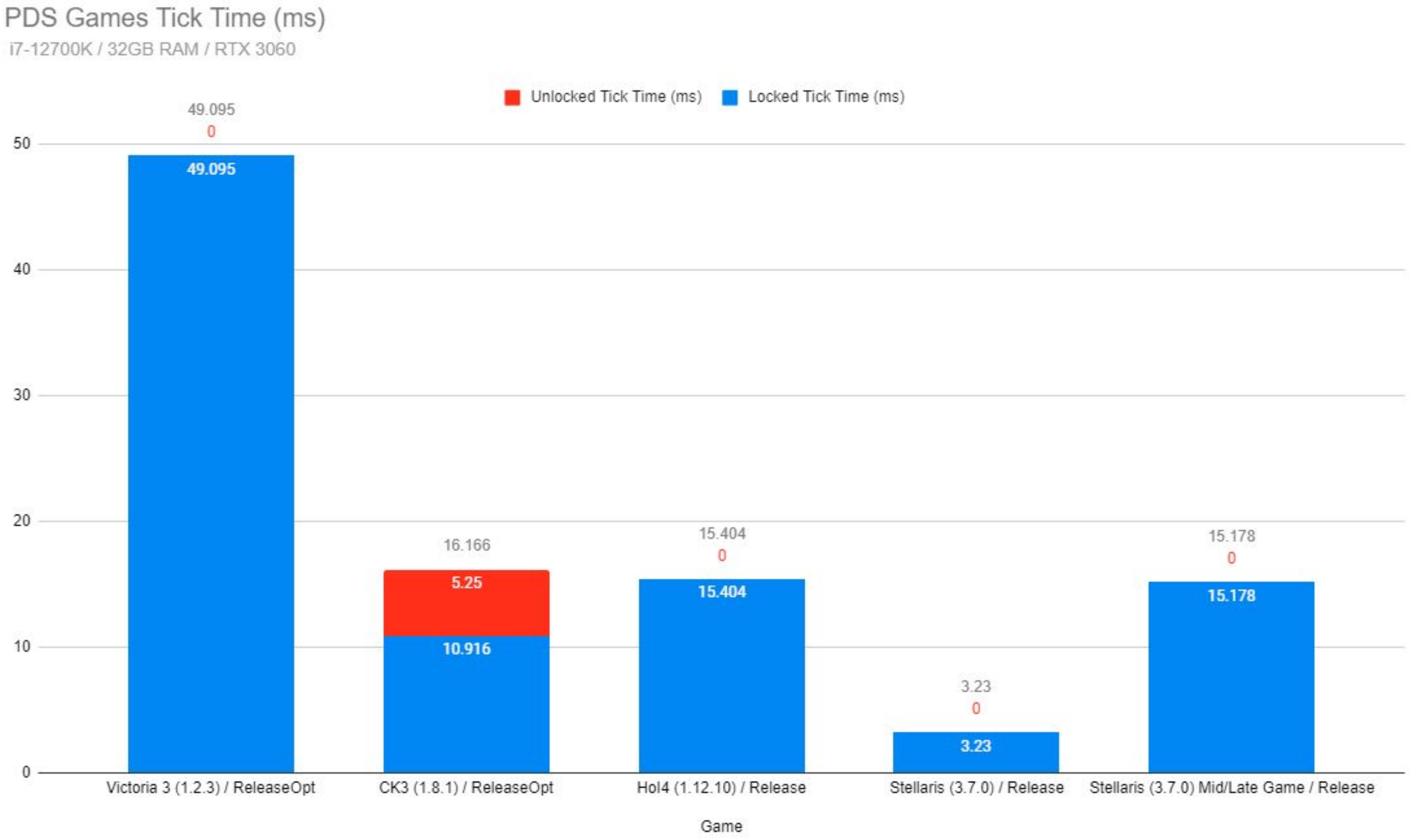


PDS Releases Timeline

PDS Games Frame Time (ms)

i7-12700K / 32GB RAM / RTX 3060




PDS Games Tick Time (ms)
i7-12700K / 32GB RAM / RTX 3060
🟥 Unlocked Tick Time (ms)
🟦 Locked Tick Time (ms)
0
10
20
30
40
50
49.095
0
49.095
16.166
5.25
10.916
15.404
0
15.404
3.23
0
3.23
15.178
0
15.178
Victoria 3 (1.2.3) / ReleaseOpt
CK3 (1.8.1) / ReleaseOpt
Hol4 (1.12.10) / Release
Stellaris (3.7.0) / Release
Stellaris (3.7.0) Mid/Late Game / Release
Game

PDS Development History

- All games use the same in-house engine, dubbed *Clausewitz*

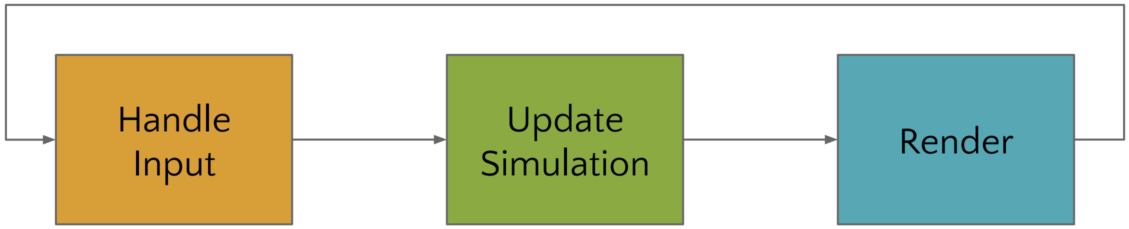
- Up until Imperator (2019), games forked the engine at some point during development

- Big generational jump between Stellaris (2016) and Imperator (2019), dubbed *Jomini*

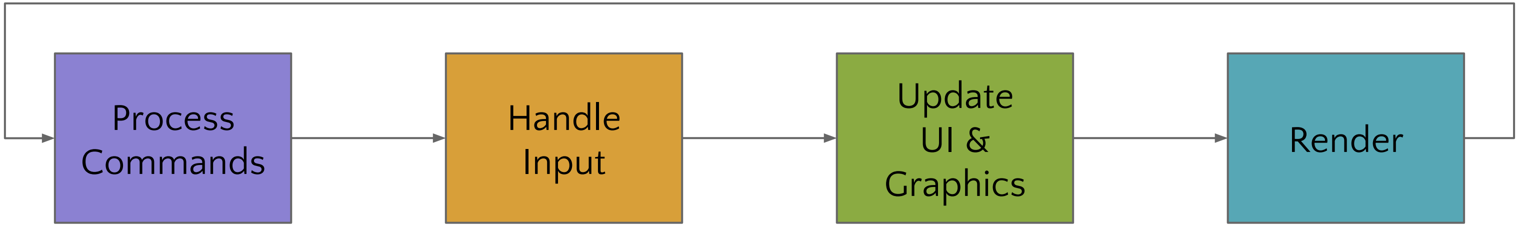
Past Generation Games

- Started with Crusader Kings II (2012)
- Multithreading done through TBB
- Mostly focused on speeding up the world simulation

Demo Time!



A Basic Game Loop



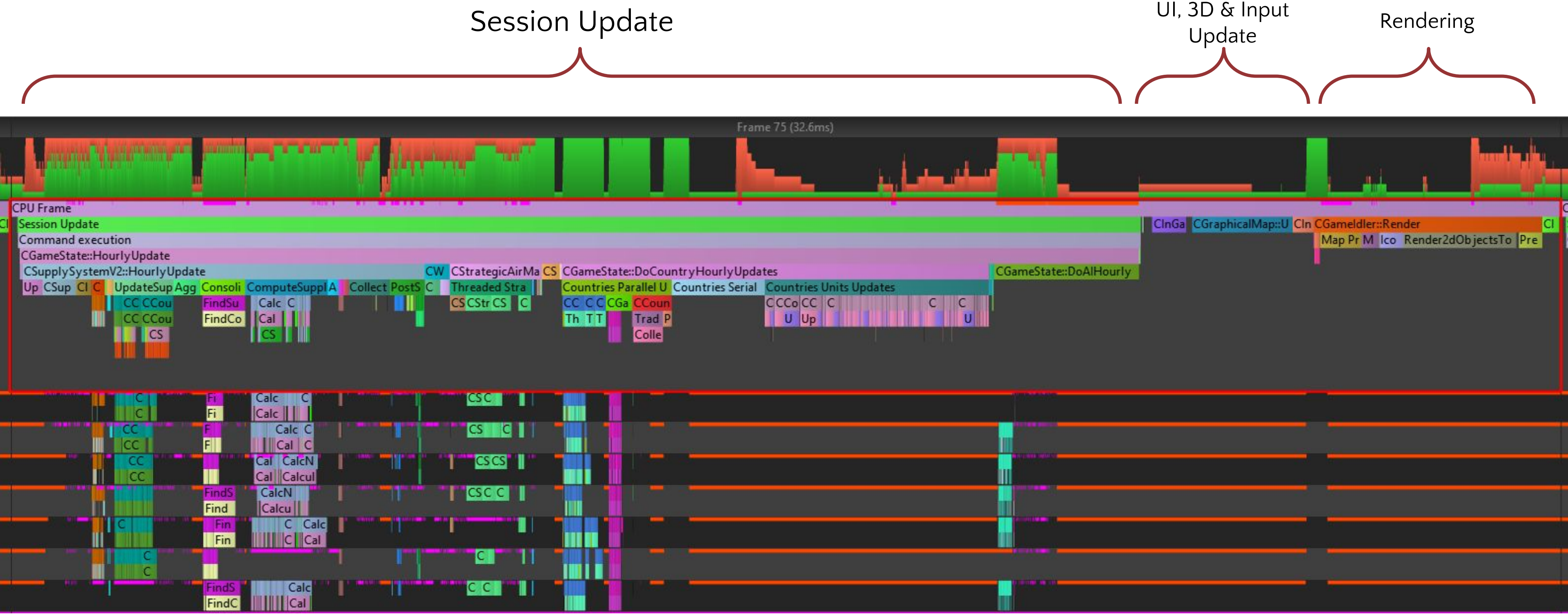
Past PDS Games Loop

Commands & Time Simulation

- Gamestate can only be changed through command execution
- Player interactions with UI result in new commands being added to the queue
- Server queues a command to advance time by one unit at real time intervals

Commands & Time Simulation

- Passage of time is simulated by “tick” increments.
- Depending on game, “ticks” can be an ingame hour (HoI4), day (CK2 and EU4) or fraction of day (Stellaris).
- No in-between!


Session Update

UI, 3D & Input Update

Rendering

Hearts of Iron IV Sample Update

Core Utilisation

- Each sub-system update is run in a sequence
- Core utilisation depends entirely on how a given system is implemented
- Most multithreading is done through `parallel_for()`

Core Utilisation

- Rule of a thumb: more recent systems have better threading efficiency
- Some have been retrofitted over the years to use parallelization
- UI & Graphics update / rendering are not done in a dedicated thread

Core Utilisation

- Board game heritage can still be felt in some game systems
- Unit/Combat update rely on sequence to be deterministic
- Hard to address in an existing game

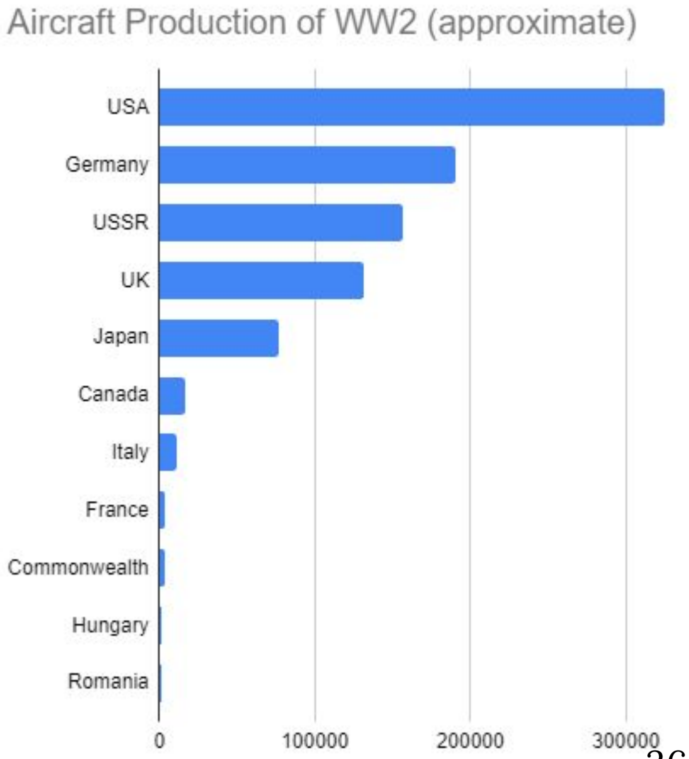
Model Limitations

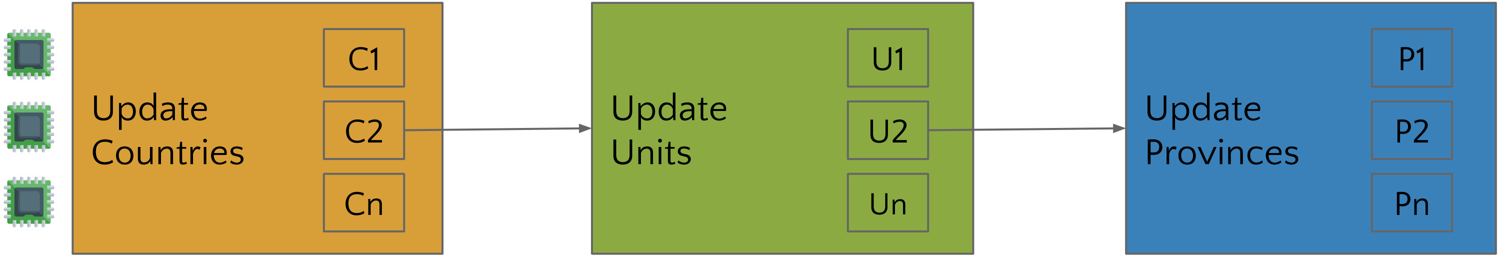
- Parallelizing updates by sub-system is limited by entity grain size

- HoI4 has:
  - 13236 provinces (tiles on map)
  - 835 states
  - 295 countries

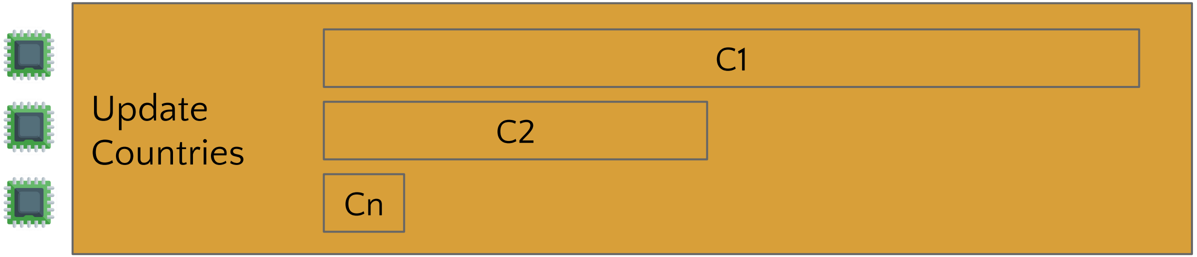
Model Limitations

- Not all entities are created equal
- Custom scheduler can help a little
- Some optimizations turn out to be pessimizations





Past PDS Games Loop



Update Countries

C1

C2

Cn

Past PDS Games Loop

Model Limitations

- Entities within a system are not equal
- System update will be as fast as the slowest entity to update, even with many cores
- Large entities in one system tend to also be big in other systems using the same breakdown

Past Model Summary

- Good enough for the time
- Some systems manage to utilize all cores
- Refitting older systems can be difficult and risky unless willing revisit game design

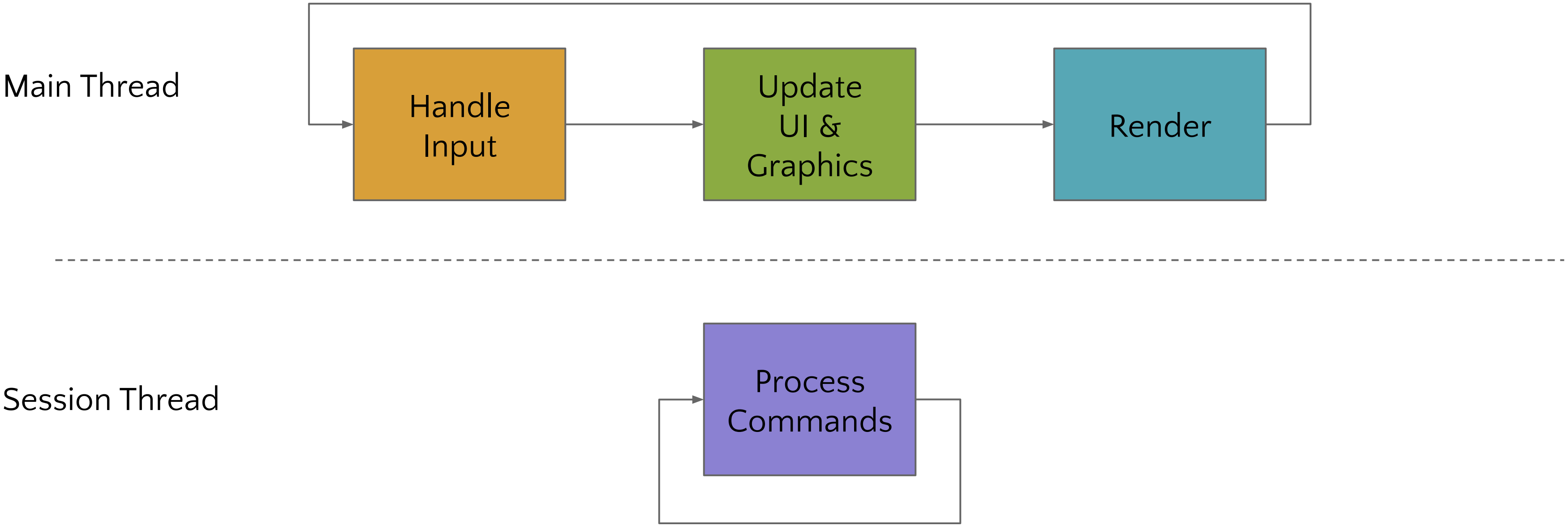
3

Multithreading in Current Generation

Illustrated mostly by Crusader Kings 3

Current Generation Changes

- At some point both Imperator, CK3 and Victoria 3 were in development simultaneously
- Same engine, different approaches to simulation update



Main Thread

Handle Input

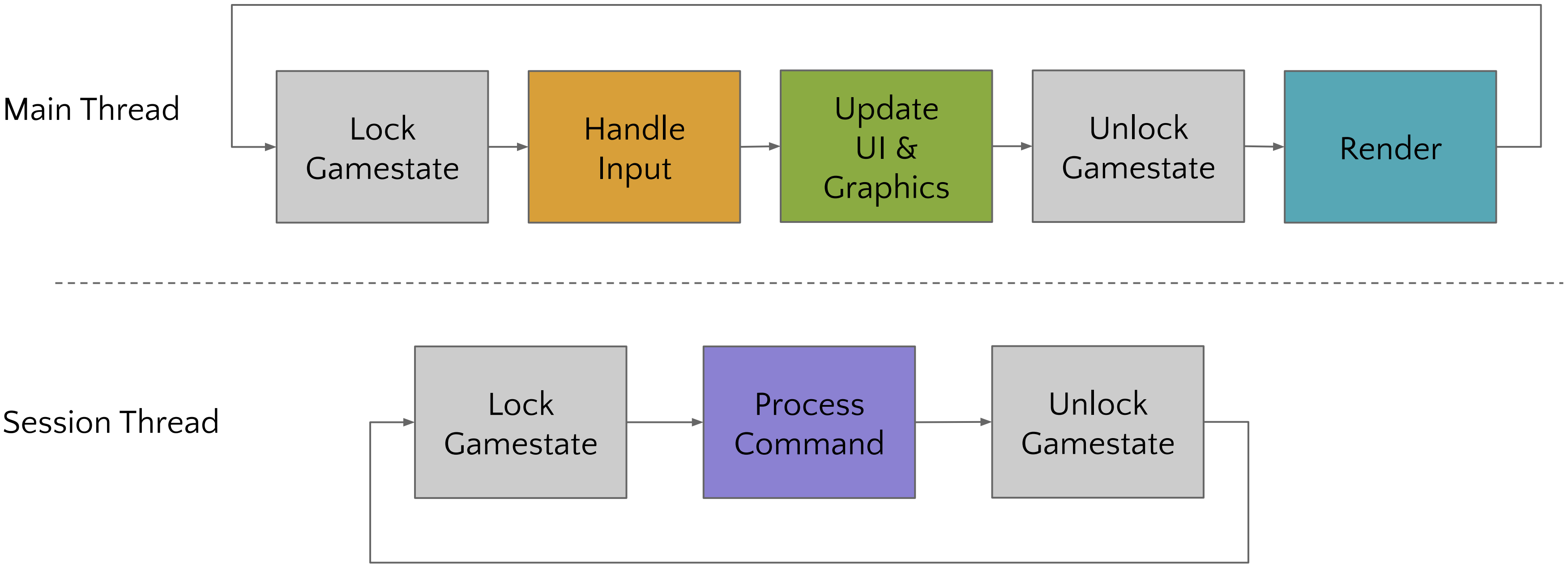
Update UI & Graphics

Render

Session Thread

Process Commands

Present PDS Games Loop

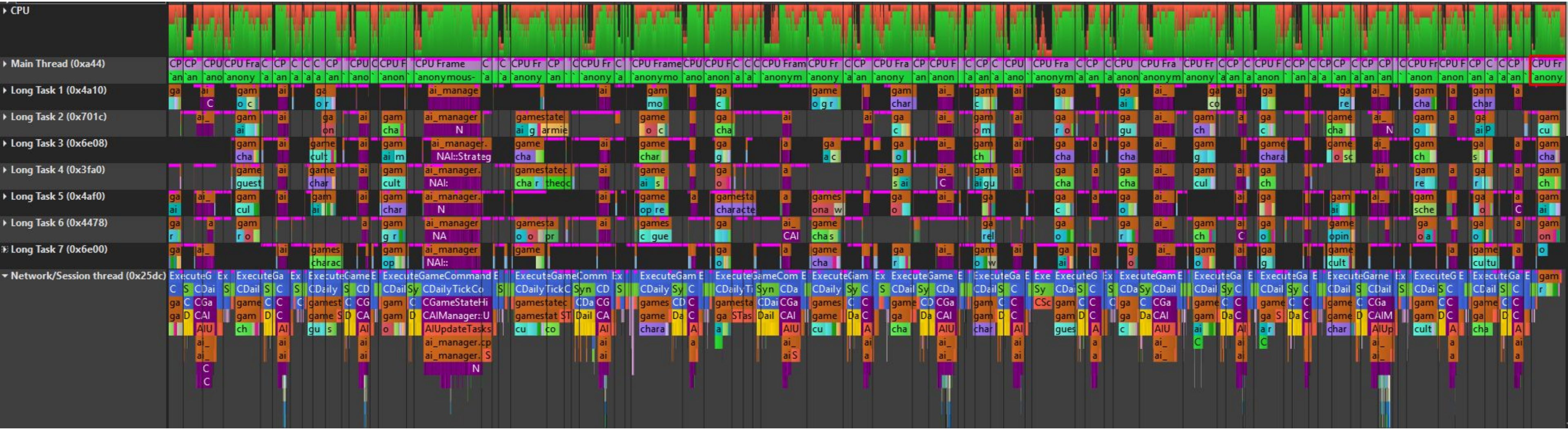


Present PDS Games Loop

Threading Efficiency

- Dedicated render thread guarantees at least some degree of multithreading
- Doesn’t solve the biggest CPU bottleneck (gamestate update) out of the box
- Mutexes 😢

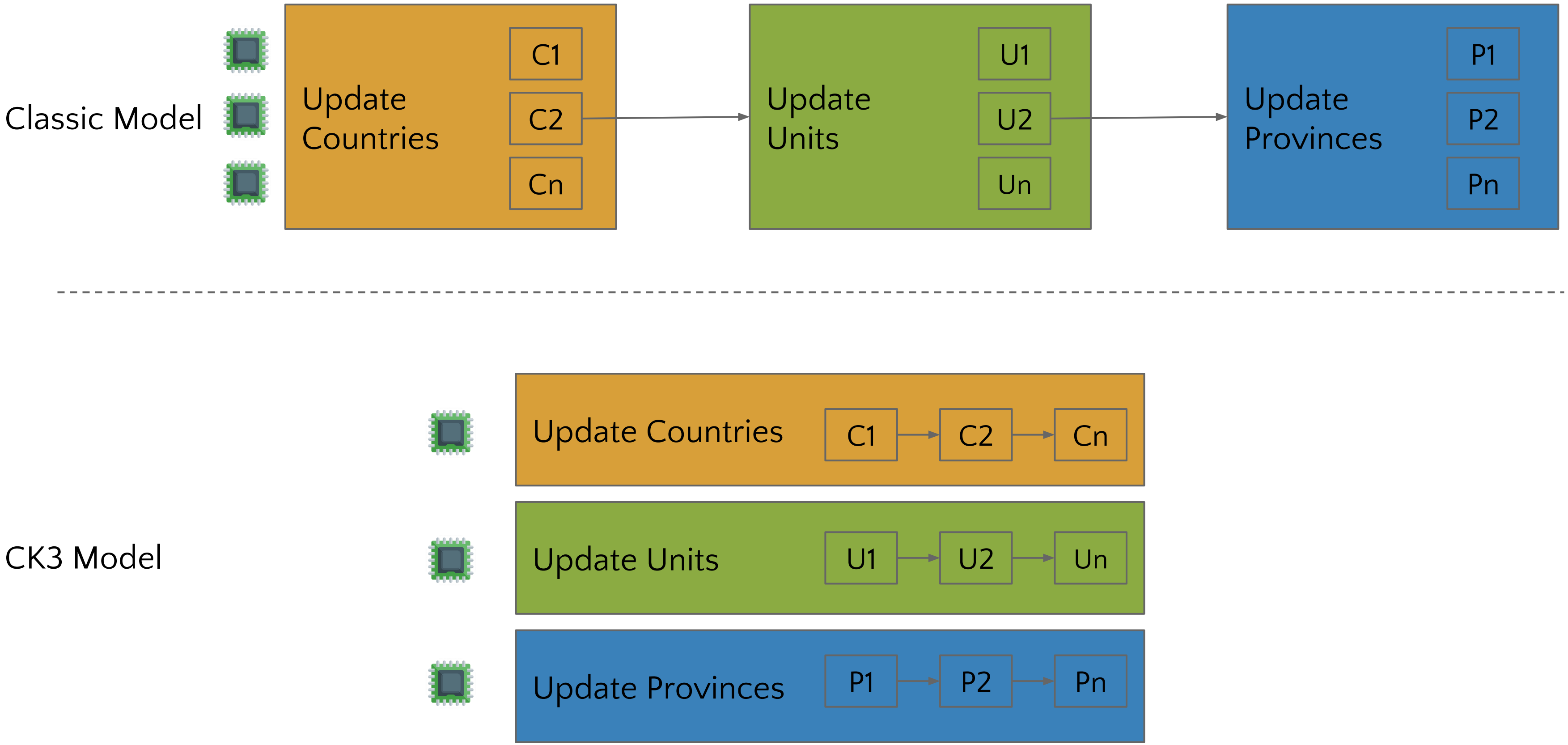
Demo Time!



Crusader Kings 3 Profile

Crusader Kings 3 Model

- Parallelize updates by system instead of by entity
- Split updates between bits that needs read and write access to gamestate
- Do the heavy lifting with only the read lock if possible



Classic Model

Update Countries

C1

C2

Cn

Update Units

U1

U2

Un

Update Provinces

P1

P2

Pn

CK3 Model

Update Countries

C1

C2

Cn

Update Units

U1

U2

Un

Update Provinces

P1

P2

Pn

Past vs Present PDS Games Loop

CK3 Game Update Principles

- Read-lock part of update can only modify "private" data in the gamestate
- Write-lock part of an update can change any data in the gamestate
- Try to keep most of the update in the first part

CK3 Update Model **Benefits**

- Entities within a system are guaranteed to be updated in a deterministic sequential order
- Read-lock part of a system update can be parallelized with other systems updates
  - And Input update, Graphics update and rendering

CK3 Update Model Benefits

- Experience has shown this model is easy to teach to newcomers
- Explain the constraints of the 2 update steps
- Newly added system immediately benefit from multithreading performance

Other Games

- Architecture-wise, the CK3 model is the one with the most potential
- Imperator used an update model fairly similar to the previous generation
- Victoria 3 tick is a series of tasks, but it can’t run them in parallel 😢

Present Model Summary

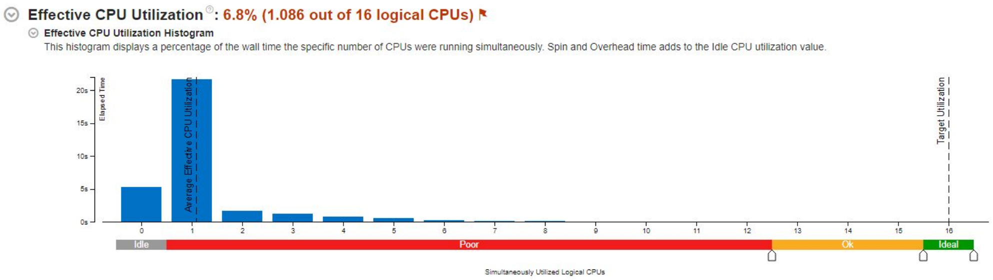
- Better threading efficiency even when combined with old school update patterns
    - Dedicated session thread 😎
- CK3 is really fast
- Model is easy enough to teach, but not enforced by the engine API

4

Thoughts for the Future

Where I look anxiously look at my NDA

HEARTS OF IRON IV

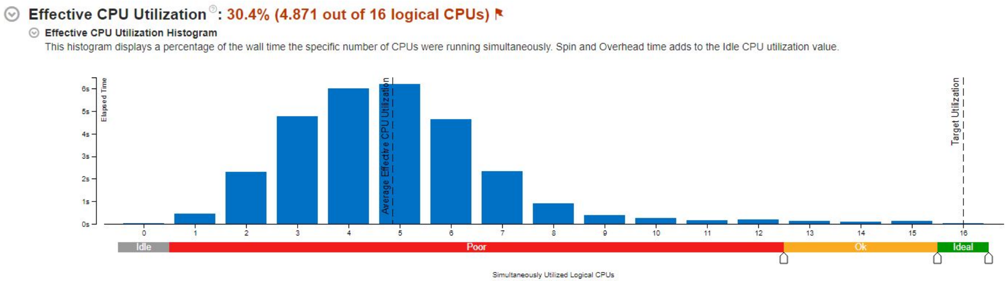


Effective CPU Utilization: 6.8% (1.086 out of 16 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

CRUSADER KINGS III



Effective CPU Utilization: 30.4% (4.871 out of 16 logical CPUs)

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

Generational Gap

Base Thoughts

- CK3 model has proven to be quite more effective than the others
- No immediate limitation to solve
- Focus on making it more accessible as a design pattern

CK3 Design Pattern

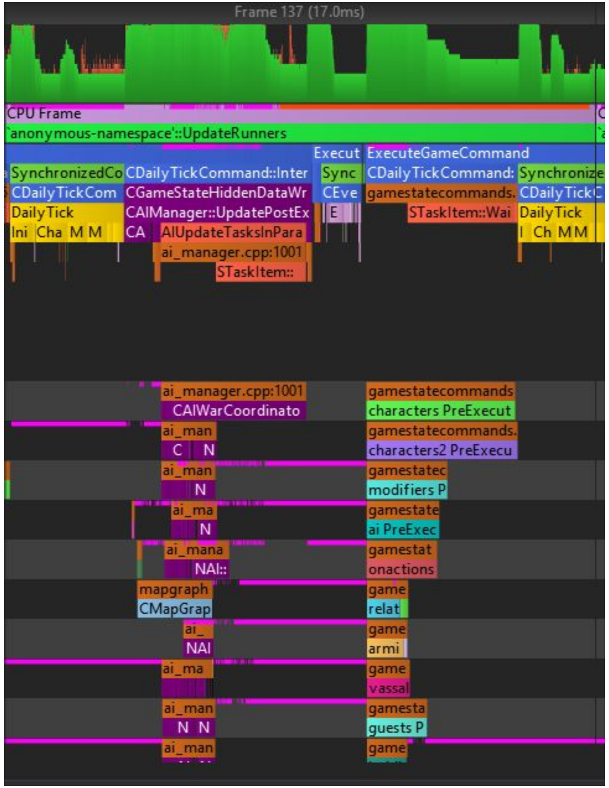
- Not formalized / enforced by the update API
- More of a best-practice to teach each time
- Implementation split between game and engine

Generalizing CK3 Pattern

- Try the same model in next title
- Move the base update model to the engine
- Rename/refactor interface to emphasize the read-lock vs write-lock update steps

Beyond the CK3 Model

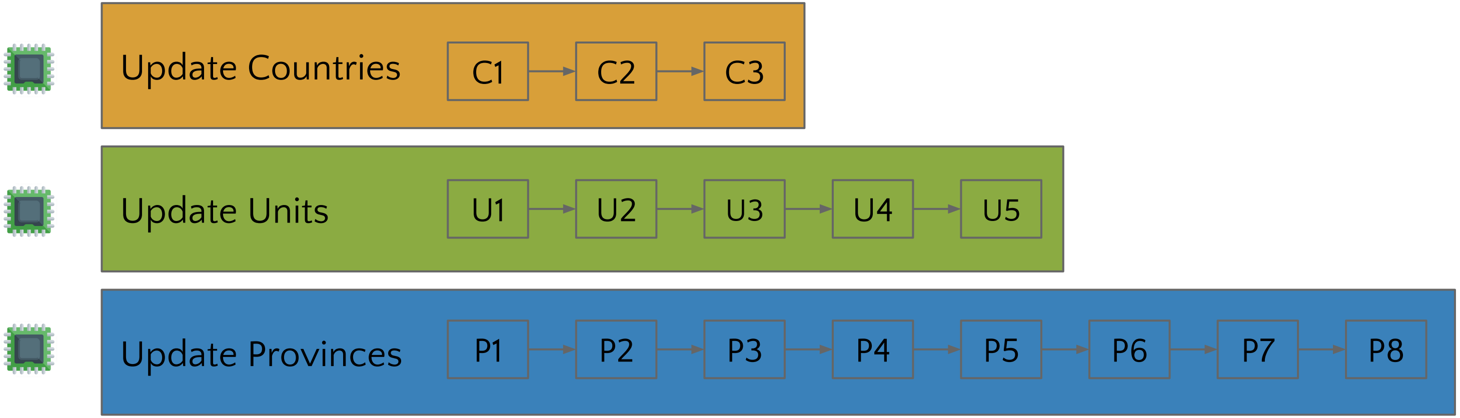
- Look at potential limitations or future hindrances
- Current average CPU utilisation on CK3 is around 5 out of 16 cores
- Can we do better?



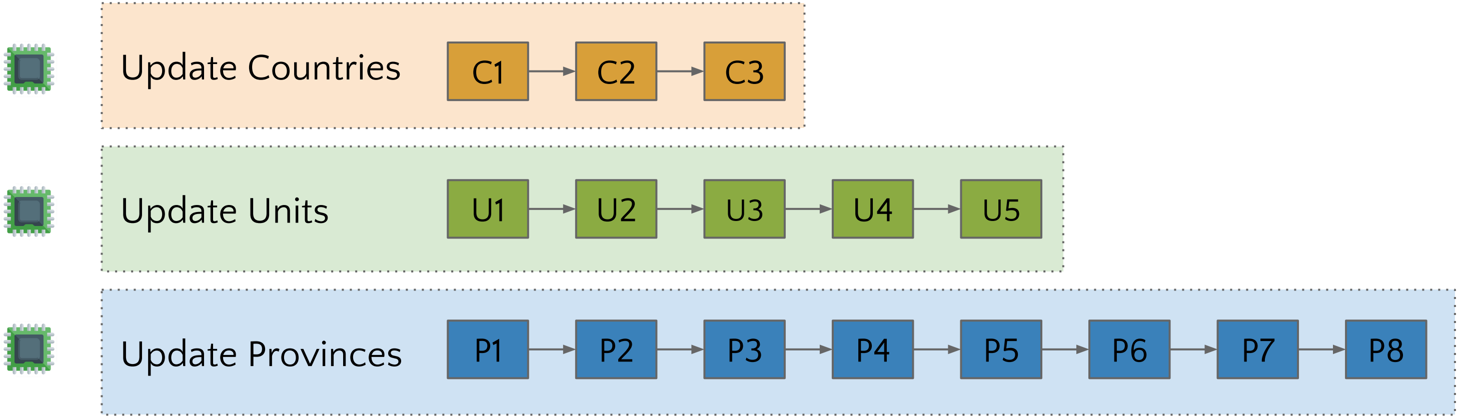
CK3 on 16 cores

Beyond the CK3 Model

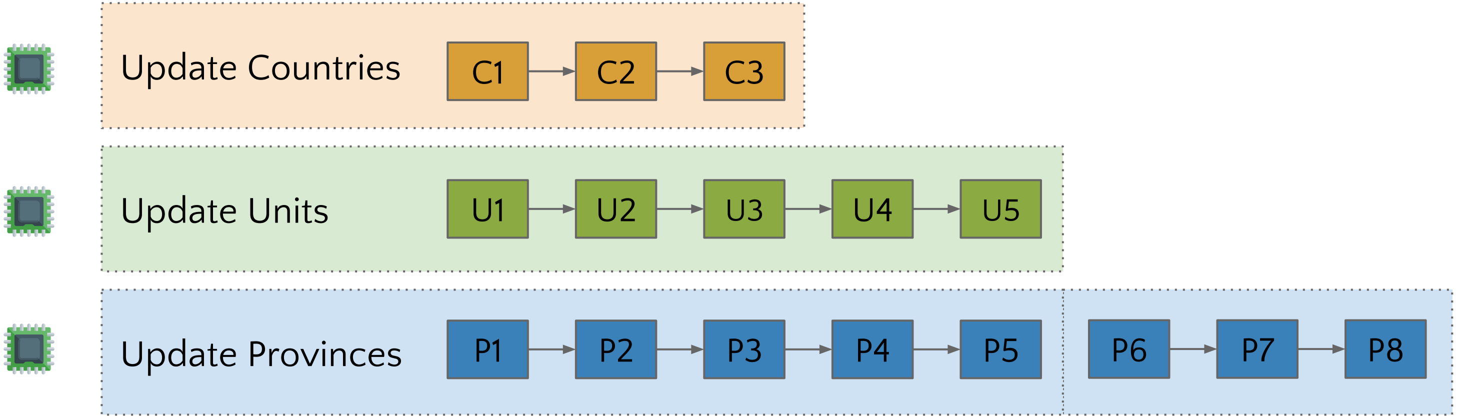
- Make entities in a system only rely on others' public data
- Have no order of execution requirement
- Each entity (or batch of entities) becomes a task you can schedule



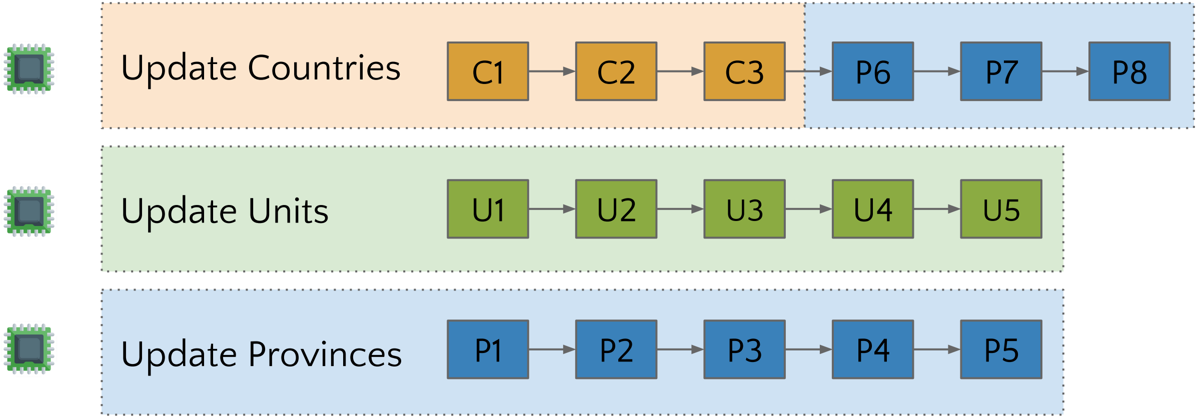
Present PDS Games Loop



Potential Future PDS Games Loop



Potential Future PDS Games Loop



Potential Future PDS Games Loop

Beyond the CK3 Model

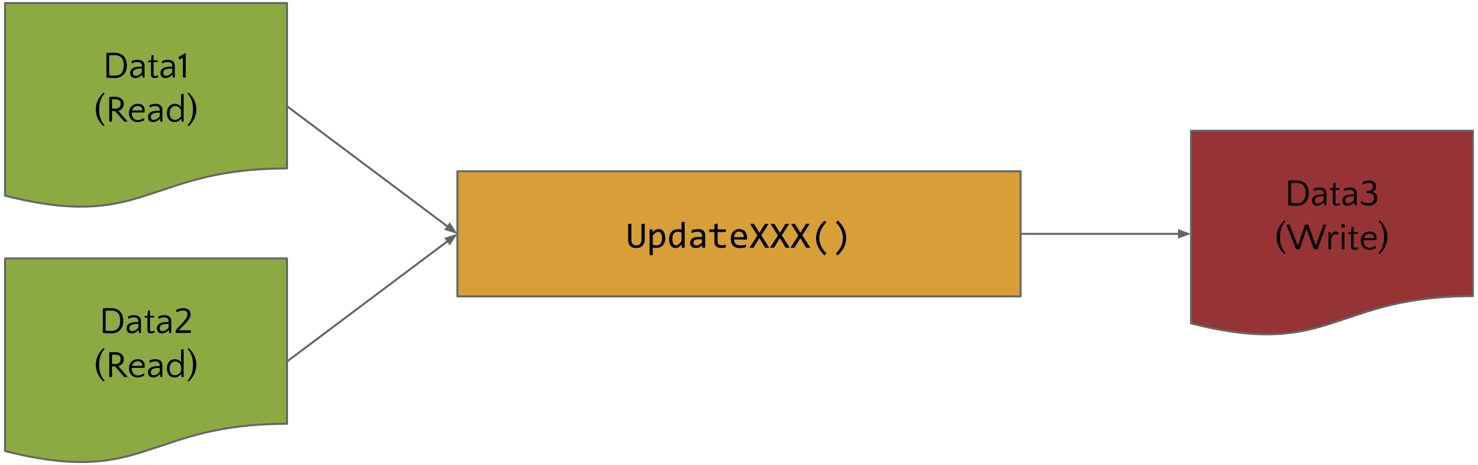
- Define sub-system update requirements
  - Read-only or read-write gamestate access
  - Entity in-order requirements

- Allow the update scheduler to break down sub-system updates into smaller chunks when the right requirements are filled

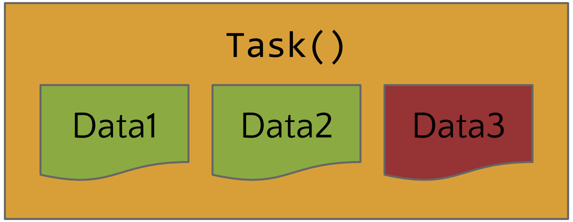
Potential Future Model

- Double-down on what CK3 started
- Make the model more explicit in the API
- Offer a way to break down systems into smaller chunks automatically if possible

Let’s take a step back



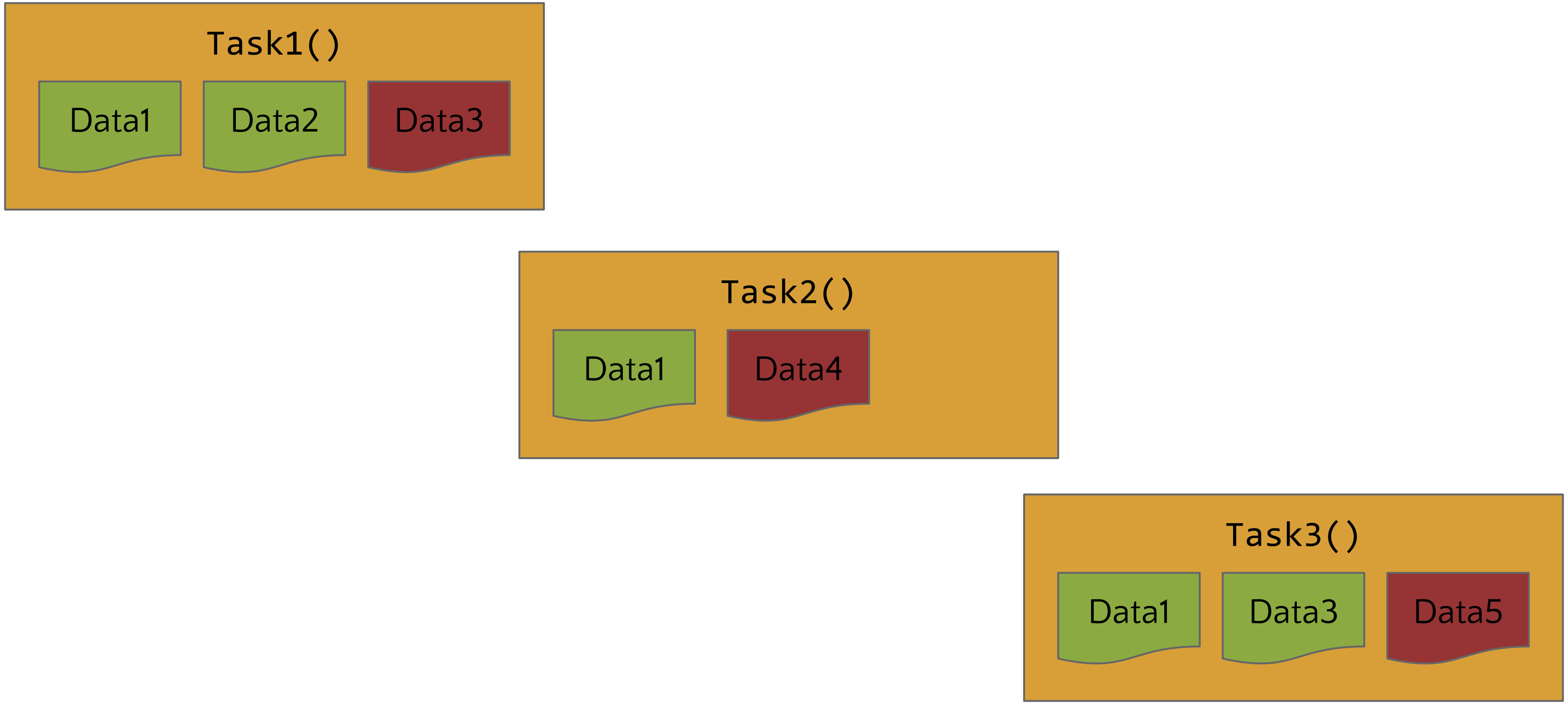
What's in a task?



What’s in a task?

`Task1()`	`Task2()`	**Concurrent?**
🟩 Data1	🟩 Data1	✅
🟩 Data1	🟥 Data1	❌
🟥 Data1	🟩 Data1	❌
🟥 Data1	🟥 Data1	❌

Concurrent Data Access



Task1()

Data1 Data2 Data3

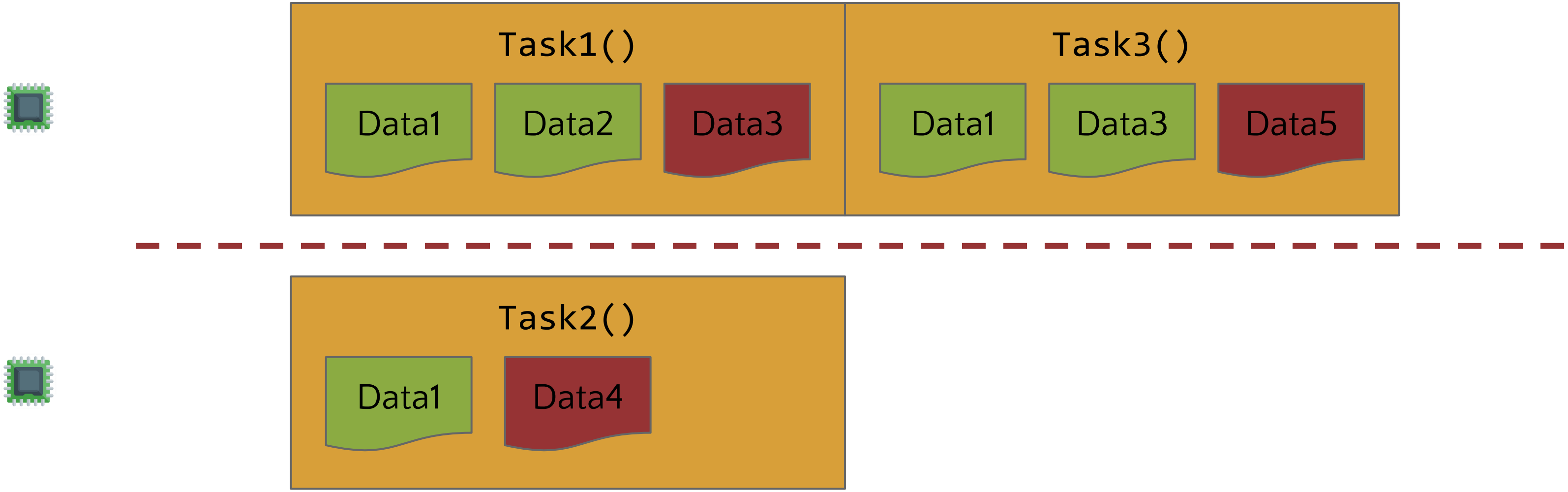
Task2()

Data1 Data4

Task3()

Data1 Data3 Data5

What's in a task?



Task1()

Data1 Data2 Data3

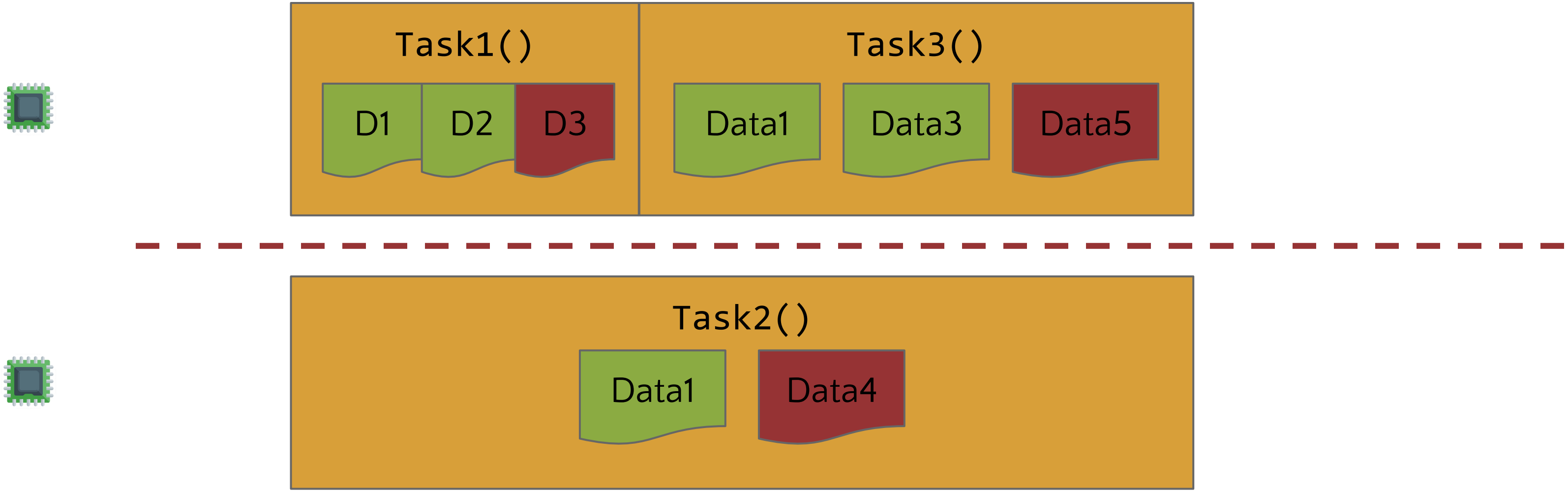
Task3()

Data1 Data3 Data5

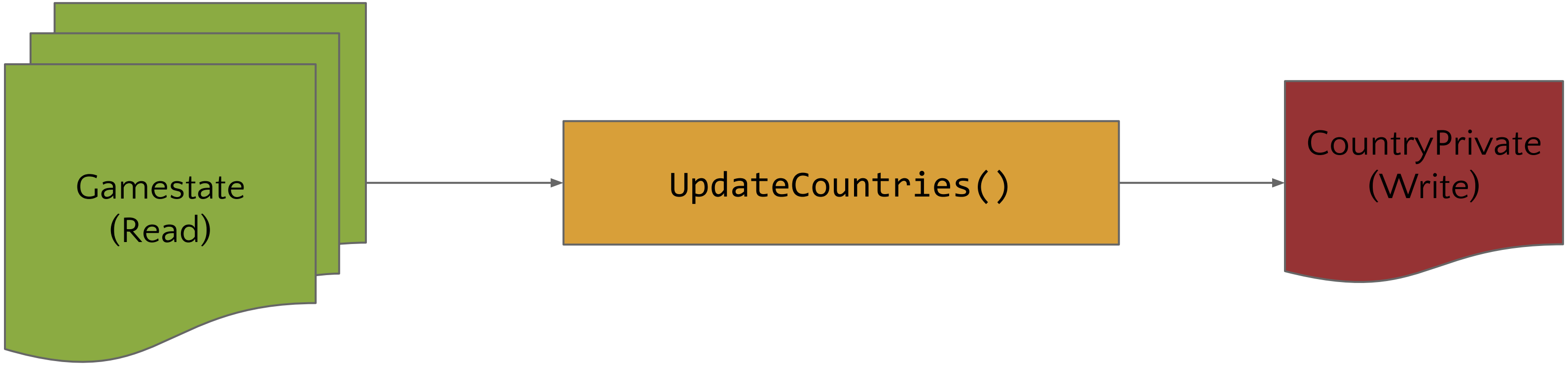
Task2()

Data1 Data4

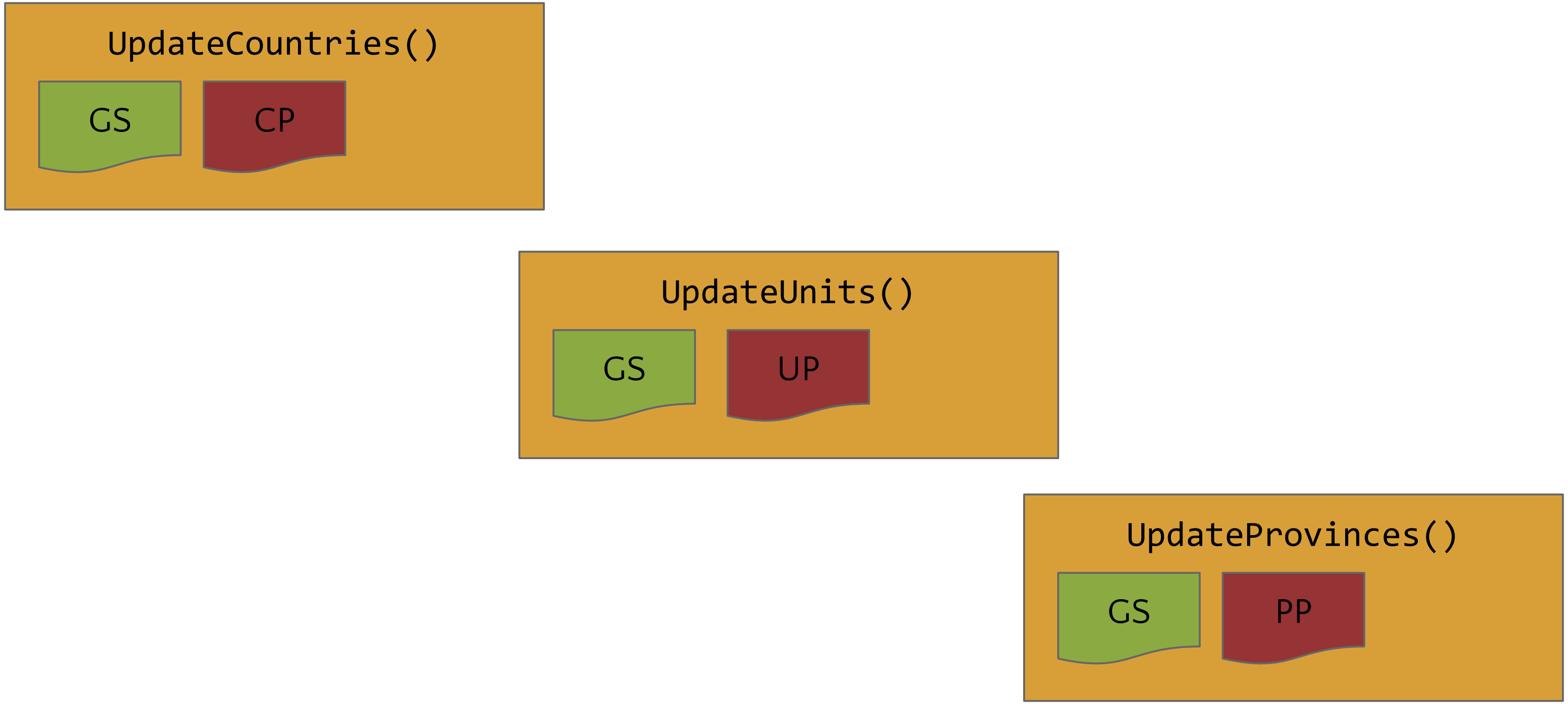
Scheduling



Scheduling



Game Tick PreUpdate Task



UpdateCountries()

GS

CP

UpdateUnits()

GS

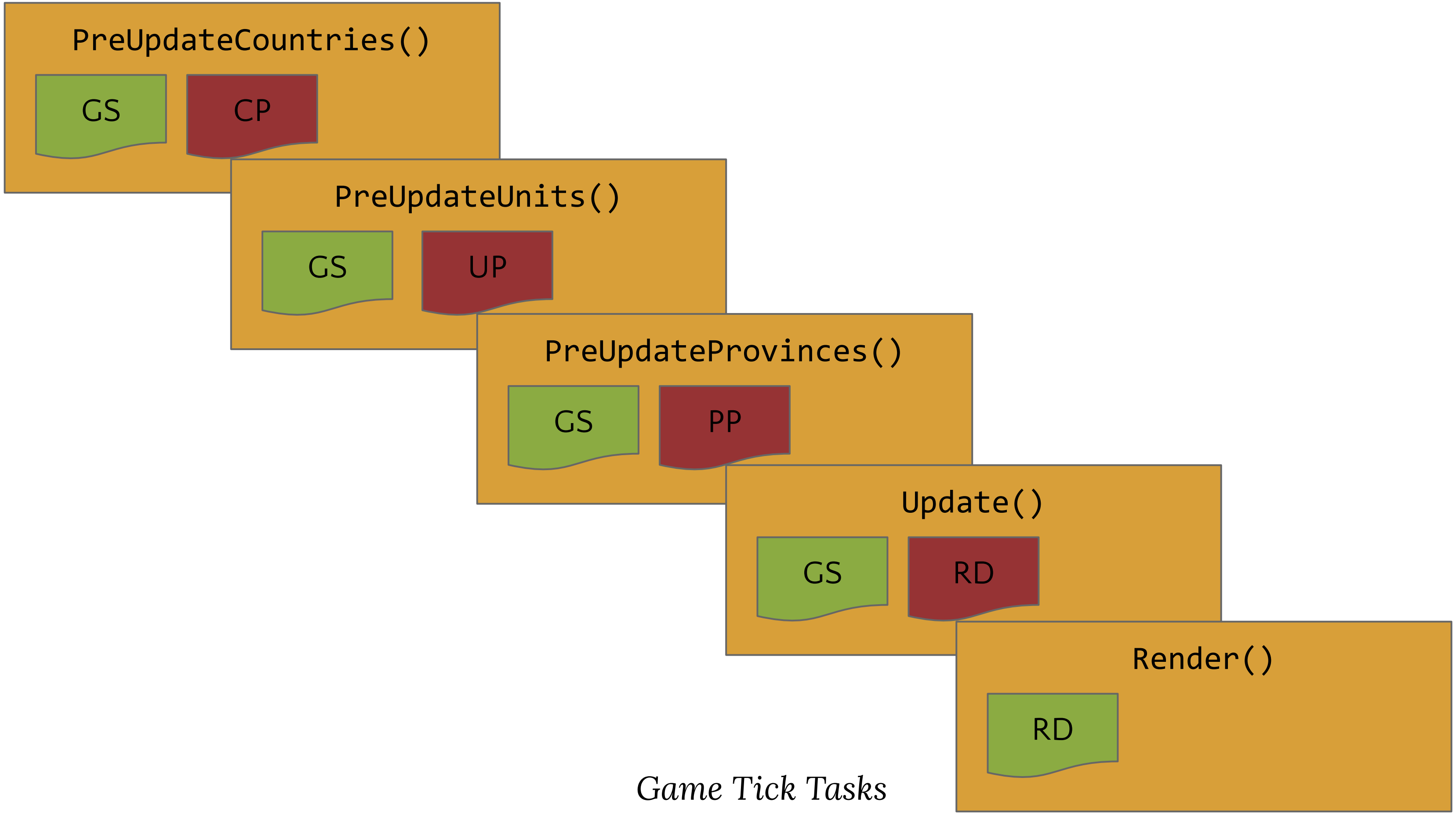
UP

UpdateProvinces()

GS

PP

Game Tick PreUpdate Tasks



PreUpdateCountries()

GS

CP

PreUpdateUnits()

GS

UP

PreUpdateProvinces()

GS

PP

Update()

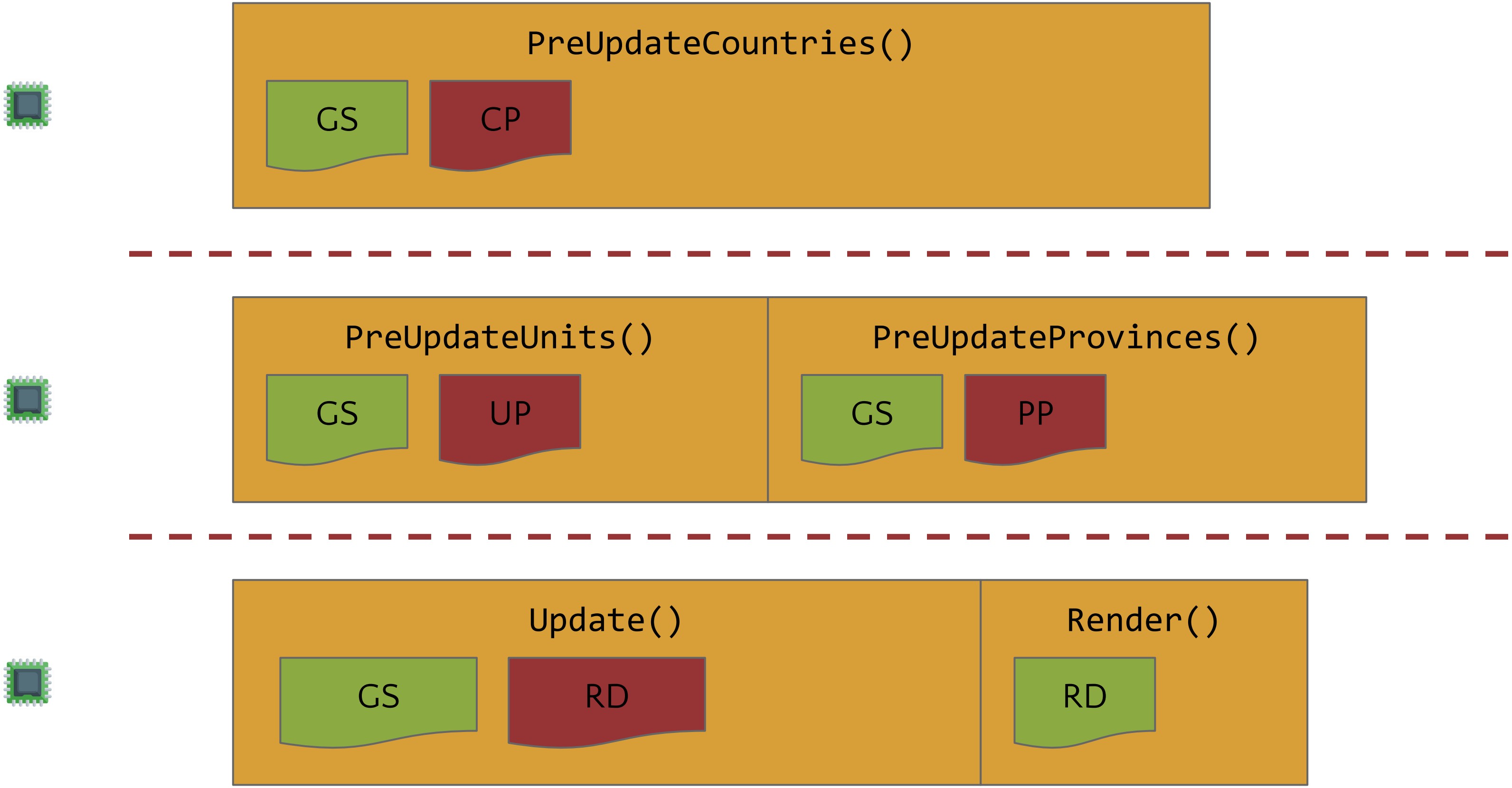
GS

RD

Render()

RD

Game Tick Tasks



Game Tick Tasks Scheduling

Potential Future Model V2

- Game Tick is a series of tasks
- Tasks have inputs (read) and outputs (writes)
- 2 tasks can be scheduled at the same time if they reads and writes don’t conflict

Potential Future Model V2

- CK3 Pre Update Tasks read all gamestate and write to a private stash
- CK3 Update Tasks read from one private stash and write to all gamestate
- Update Task reads all gamestate

Potential Future Model V2

- Better define update tasks read and writes (which subsections of the gamestate)
- Make scheduler consider both task logical dependencies and r/w data access
- Fit update/render task in that model?

Potential Future Model V2

- CK3 model is really simple to reason about and teach to new programmers
- Adding task dependencies might run against parallelism if there are too many
- Explicit data usage declaration make design iteration more expensive

5 Wrapping up

The threads of past, present and future

In conclusion

- Using modern CPUs efficiently require good core utilization

- Adding `parallel_for` to existing code only gets you so far

- Adopting a model that enforces thread-friendly constraints is key

Furthermore

Furthermore, I think your build should be destroyed

Thanks!

Any questions ?

You can reach me at

✉ mro@puchiko.net

@MatRopert

@mropert

https://mropert.github.io