Многопоточность и игры

Igor Lobanchikov, 2019

Игорь Лобанчиков

- Разрабатываю игры с 2003 года
- S.T.A.L.K.E.R.: Clear Sky
- Работал с Intel, AMD, Qualcomm, Amazon, Confetti и другие
- Эксперт по компьютерной графике
  - Помогаю улучшать и оптимизировать чужие проекты
  - Портирую игры на новые платформы
- imixerpro(at)gmail(dot)com

О чем будем говорить

- Структура application loop
- История многопоточности в играх
  - Как зарождалась
  - Как развивалась
  - Почему именно так, а не иначе
- Многопоточность сегодня
  - Task-based
  - Hybrid (Task-based + thread-based)
- Многопоточный рендеринг
  - DX12 + Vulkan + Metal

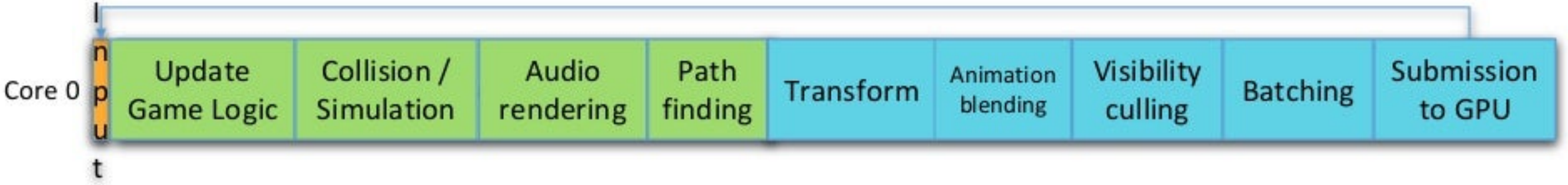
Структура application loop

Структура application loop

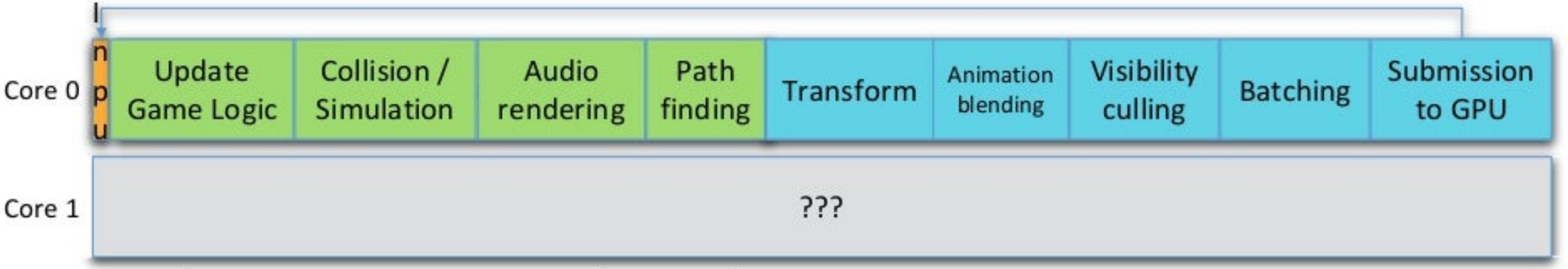
- 60 кадров в секунду (16.6 мс на кадр)
  - 16.6мс vs 17.6мс = 60 FPS vs 57 FPS
  - 33.3мс vs 34.4мс = 30 FPS vs 29 FPS

Структура application loop

Структура application loop

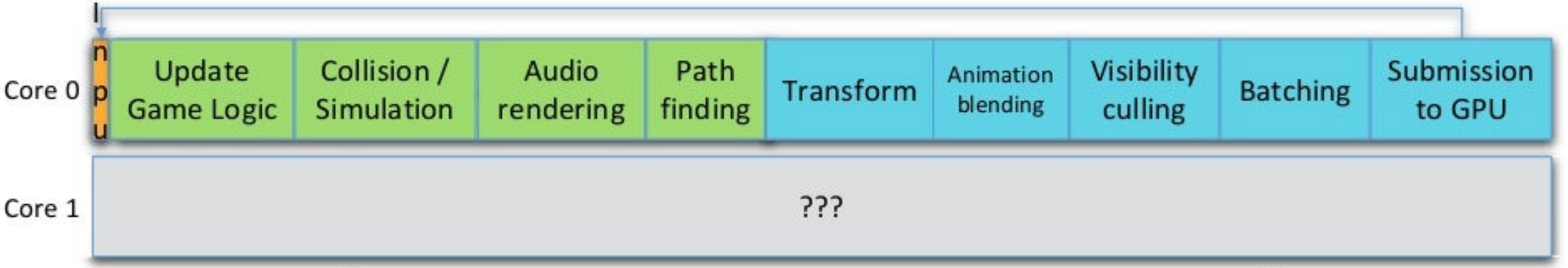


Предпосылки возникновения многопоточности



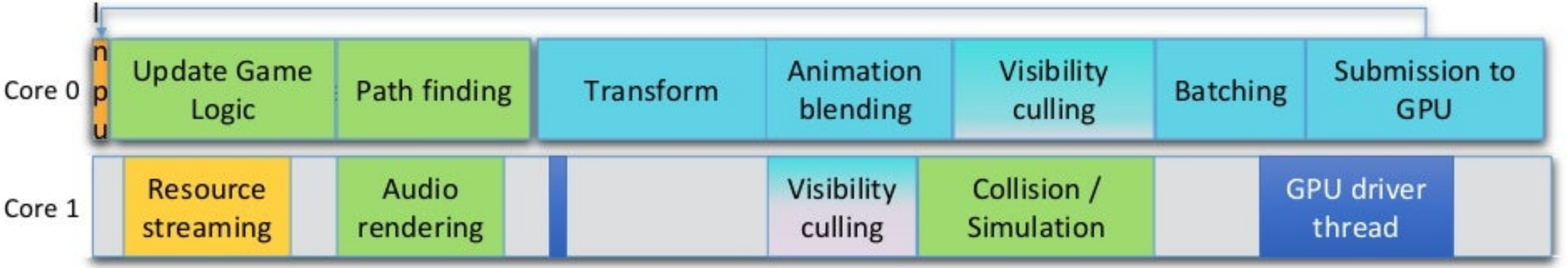
- February 25, 2002, Xeon (2xSMT)
- Early 2003 MS announced XBox Next (3 cores w/2xSMT), launched November 22, 2005
- May 25, 2005, Pentium D (2 cores)
- May 2005, Athlon 64 X2 (2 cores)
- November 11, 2006 Sony launched PS3 (1 PowerPC 2xSMT core, 6 SPEs)

Предпосылки возникновения многопоточности



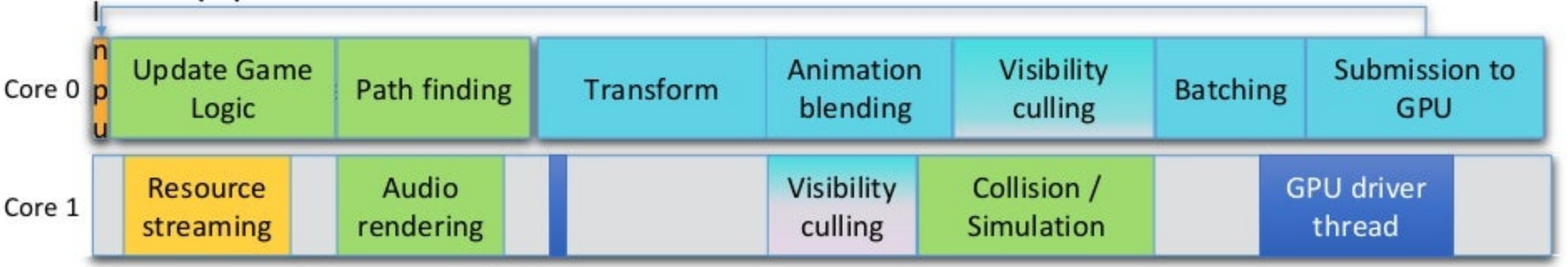
- Основной парк игровых компьютеров - одноядерный
- Двух-ядерные CPU дороги
- Логические взаимосвязи между различными модулями сильны ради увеличения производительности
- Консольное железо уникально и требует специального подхода

Threaded model: naive approach



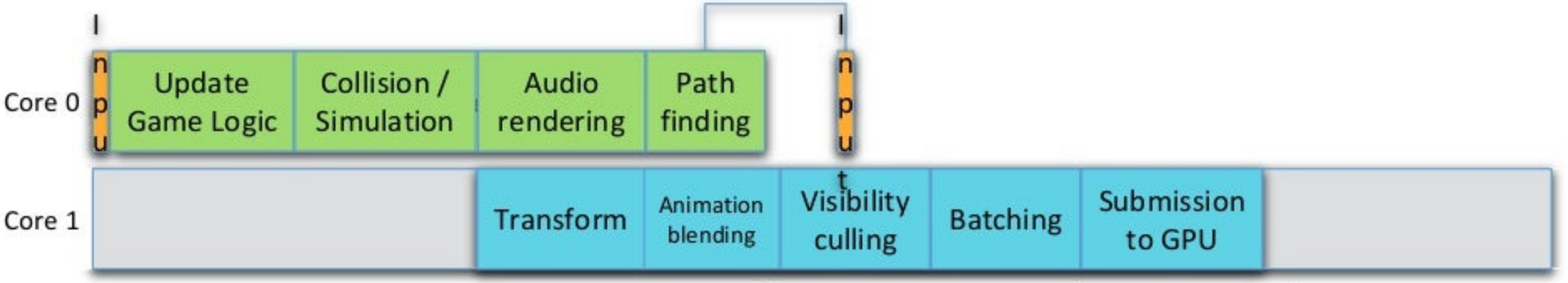
- специальный отдельный поток с фиксированным порядком выполнения задач, определенных на этапе компиляции
- минимально необходимые изменения в структурах данных
- максимально сохраняется порядок выполнения
- синхронизация с помощью conditional или их аналогов
- возможно, у каждой задачи свой поток, поскольку задач мало

Threaded model: naive approach Недостатки



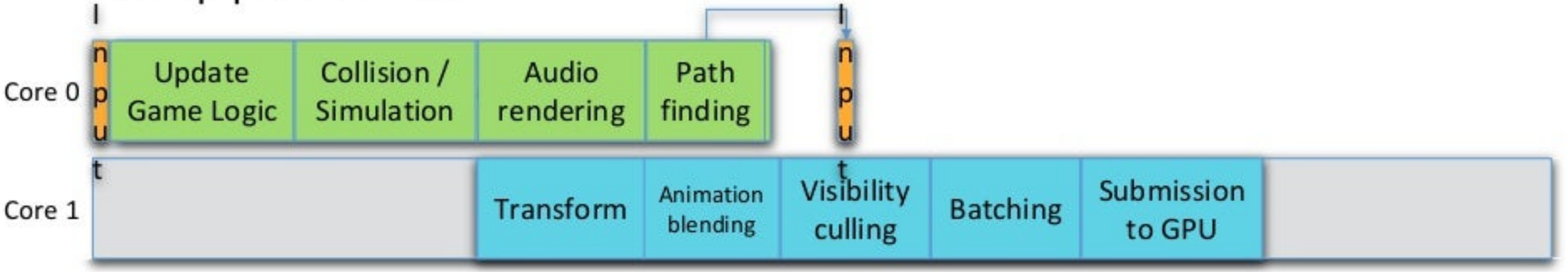
- Незначительный прирост производительности
- Плохо масштабируется при увеличении количества ядер
  - плохо подходит для XBox360/PS3

Threaded model: независимые системы



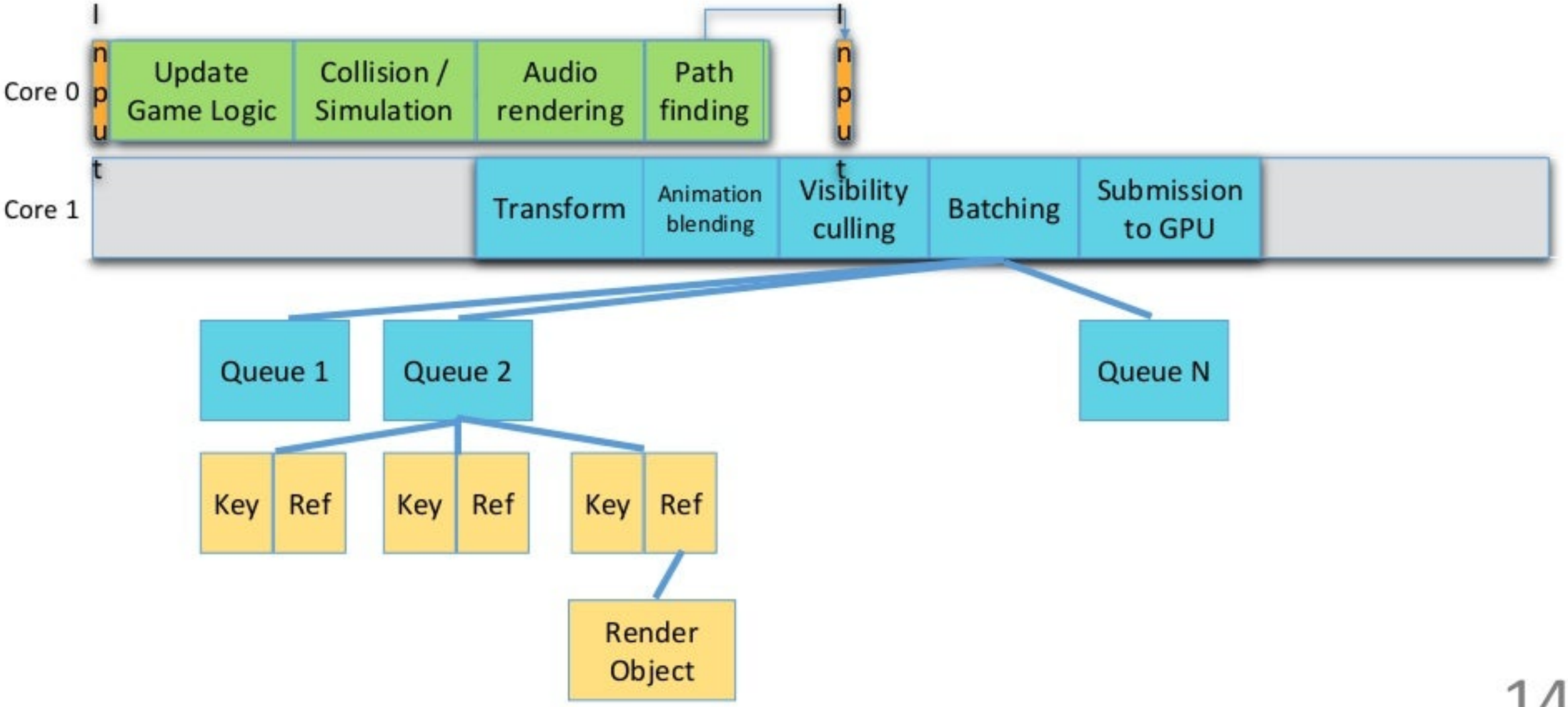
- Независимые пространства объектов для Update и Render все еще позволяют использовать старый однопоточный код
- Обновление состояния render через single writer single reader lock free кольцевой буфер
- Можно сбалансировать нагрузку для любого, заранее заданного, количества ядер - хорошо сработало для Xbox 360

Threaded model: независимые системы Недостатки

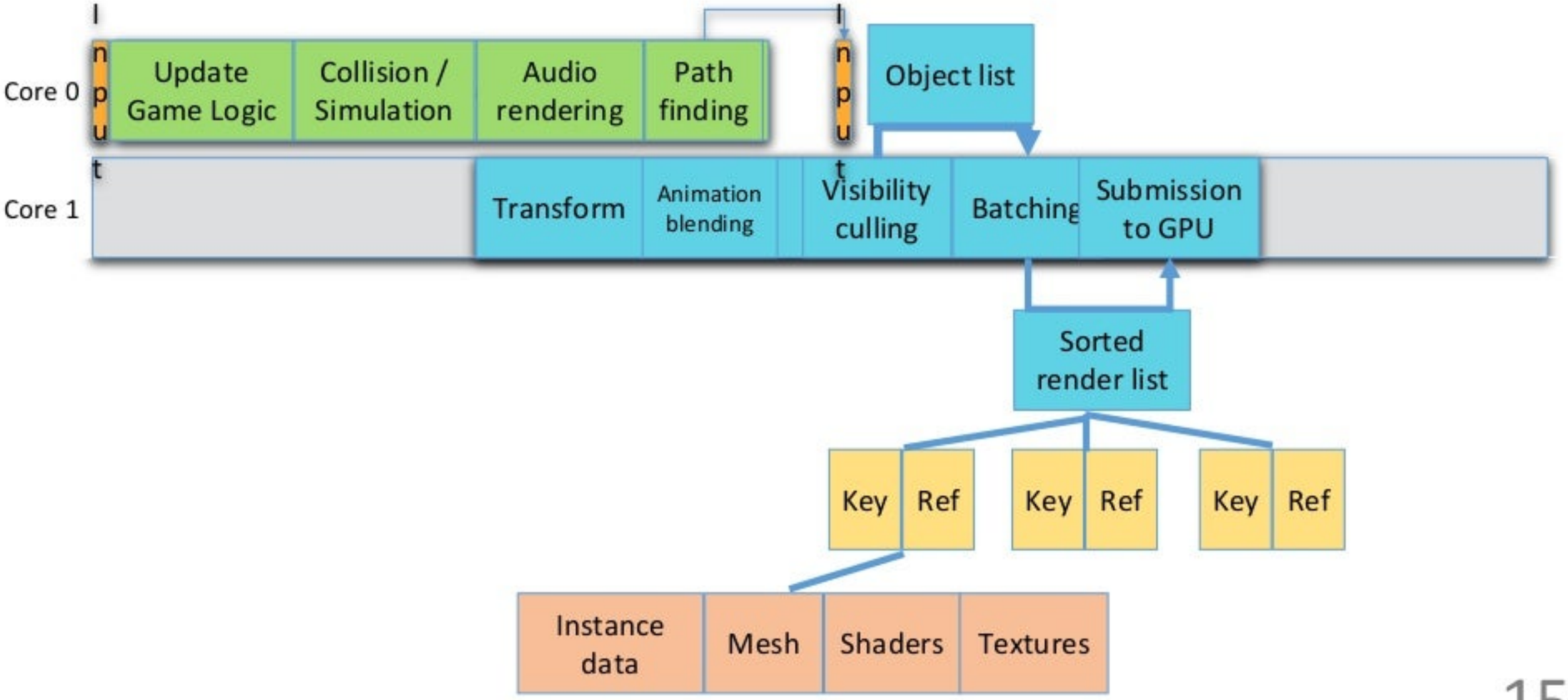


- Существенные затраты на создание независимых структур данных
- Все несколько сложнее, чем выглядит, поскольку результаты работы систем могут быть нужны в обоих потоках (например, скиннинг)
- Не масштабируется для произвольного количества ядер
- PS3 в силу наличия только одного полноценного 2xSMT ядра требовала еще более глубокой адаптации

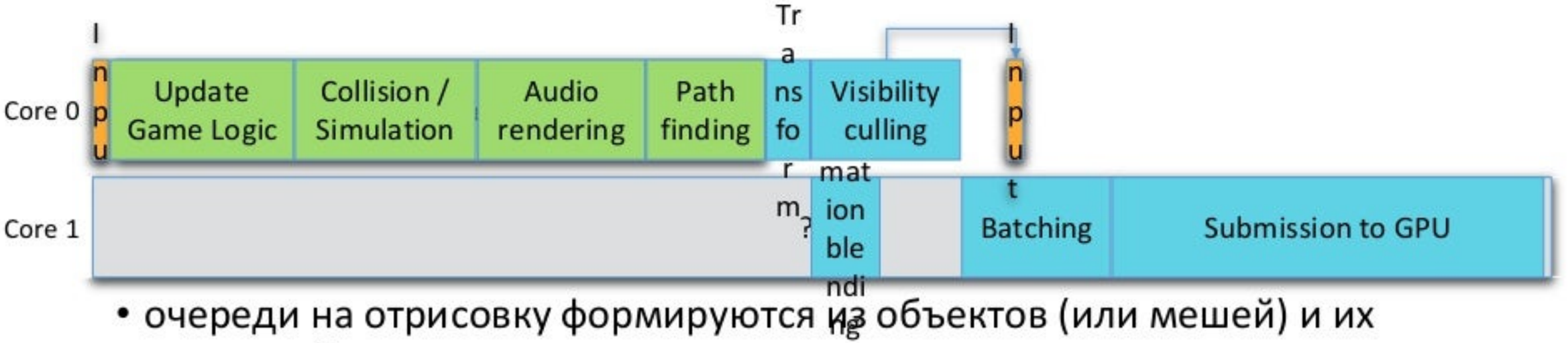
Threaded model: ???



Threaded model: ???

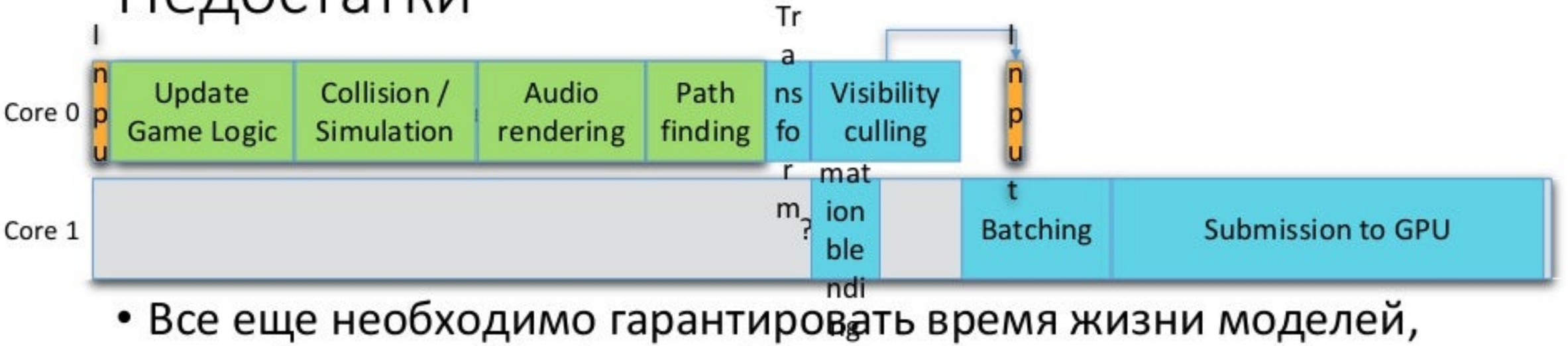


Threaded model: draw list generation



- очереди на отрисовку формируются из объектов (или мешей) и их состояний: положения в пространстве, анимации и т.п.
  - Можно пользоваться старыми однопоточными структурами данных
- Самая дорогая и неделимая часть выполняется параллельно основному потоку
- Можно сбалансировать нагрузку для любого, заранее заданного, количества ядер - хорошо сработало для Xbox 360

Threaded model: draw list generation Недостатки



- Все еще необходимо гарантировать время жизни моделей, мешей, текстур (без boost::shared_ptr)
- Выше асимметрия нагрузки потока, чем для независимых систем
- Не масштабируется для произвольного количества ядер
- те же проблемы с PS3, что и для независимых систем

Куда дальше?

- Все переписать и сделать правильно
- Data-driven approach
- Task-based approach
- C++ 17
- AGILE
- GIT
- Data science
- Block chain

Реальный мир

- Количество производственных ресурсов ограничено: люди, деньги не бесконечны
- Необходимо продолжить выпускать игры
- Количество ядер у пользователя 1-2-4 с топовыми конфигурациями 8-16
- На горизонте новые консоли (первые дев киты попали к разработчикам в 2011 году)
- Много специализированного PS3 кода для использования 6 ядер SPU

Чего мы хотим от многопоточности

- минимизация накладных расходов
  - минимум синхронизаций между потоками
- максимизация загрузки всего CPU
- минимизация конфликтов между кешами разных ядер (т.е. не пишем туда, откуда читает другой поток, и, тем более, не пишем в одно и то же место из разных потоков)

Что уже реализовано

- у нас уже есть определенные гарантии на неизменность некоторых состояний объектов и на время их жизни

Что уже реализовано

- у нас уже есть определенные гарантии на неизменность некоторых состояний объектов и на время их жизни
- мы уже умеем работать с быстрыми очередями (1R/1W)

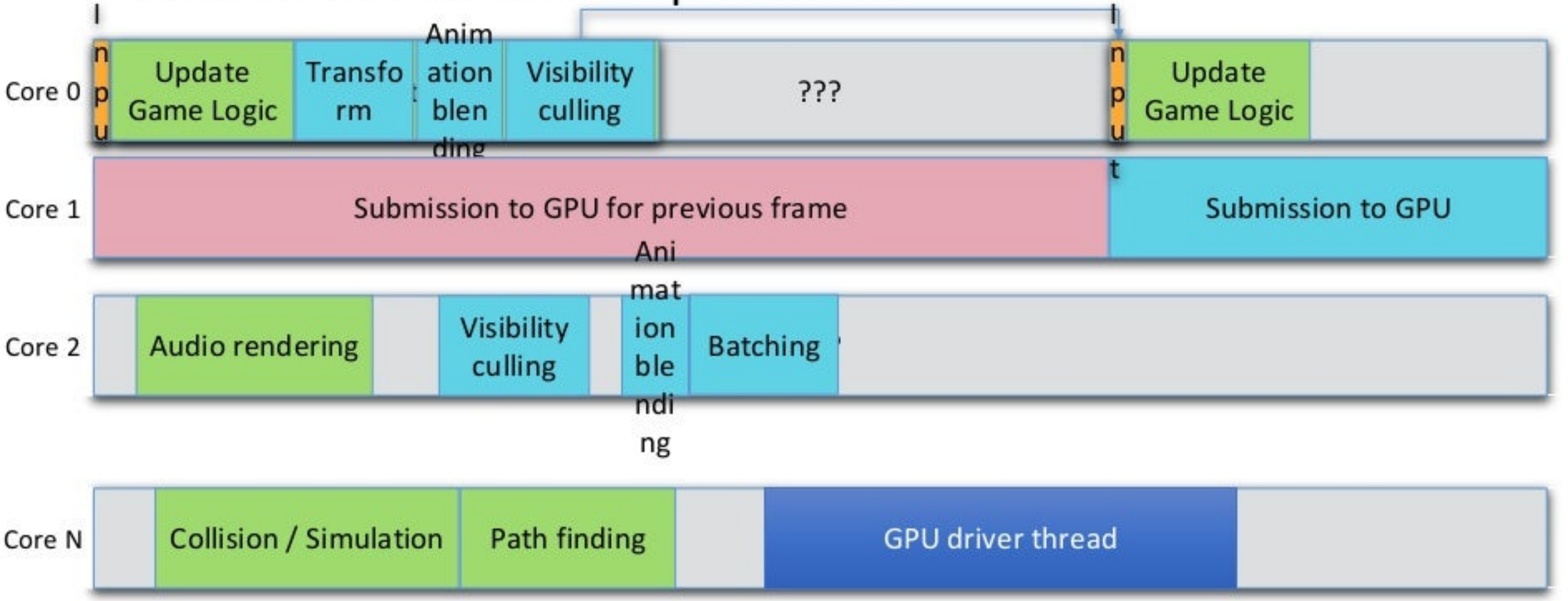
Что уже реализовано

- у нас уже есть определенные гарантии на неизменность некоторых состояний объектов и на время их жизни
- мы уже умеем работать с быстрыми очередями (1R/1W)
- Мы уже научились не выделять память как попало (до появления многопоточности тоже умели)

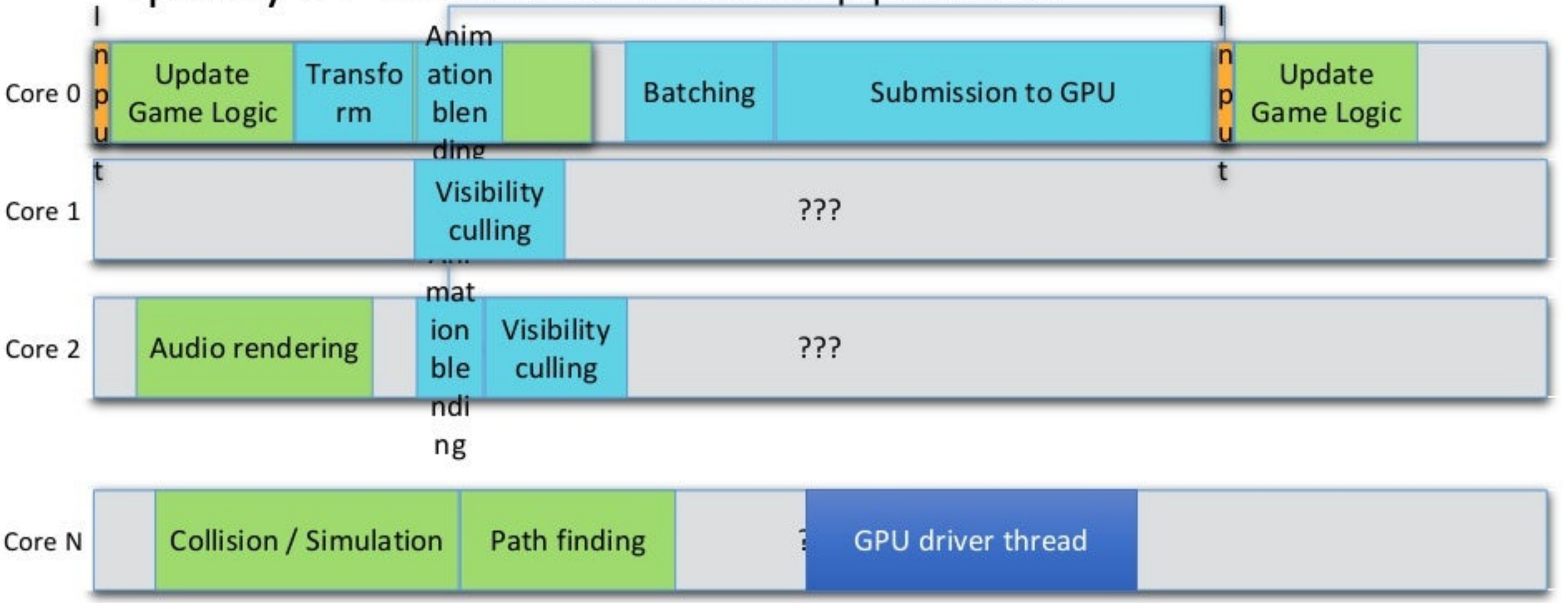
Task-based parallelism

- task manager
- выносим код из потоков в таски
- таски не модифицируют глобальные данные, но формируют новые
  - а некоторые и на вход получают буфер с данными
- используем где возможно data parallel таски (если порядок обработки не важен)
  - каждый поток таска при этом генерирует свой собственный выходной буфер (нет проблем с синхронизацией, нет проблем с кешом)
  - при необходимости по завершению работы буфера будут объединены

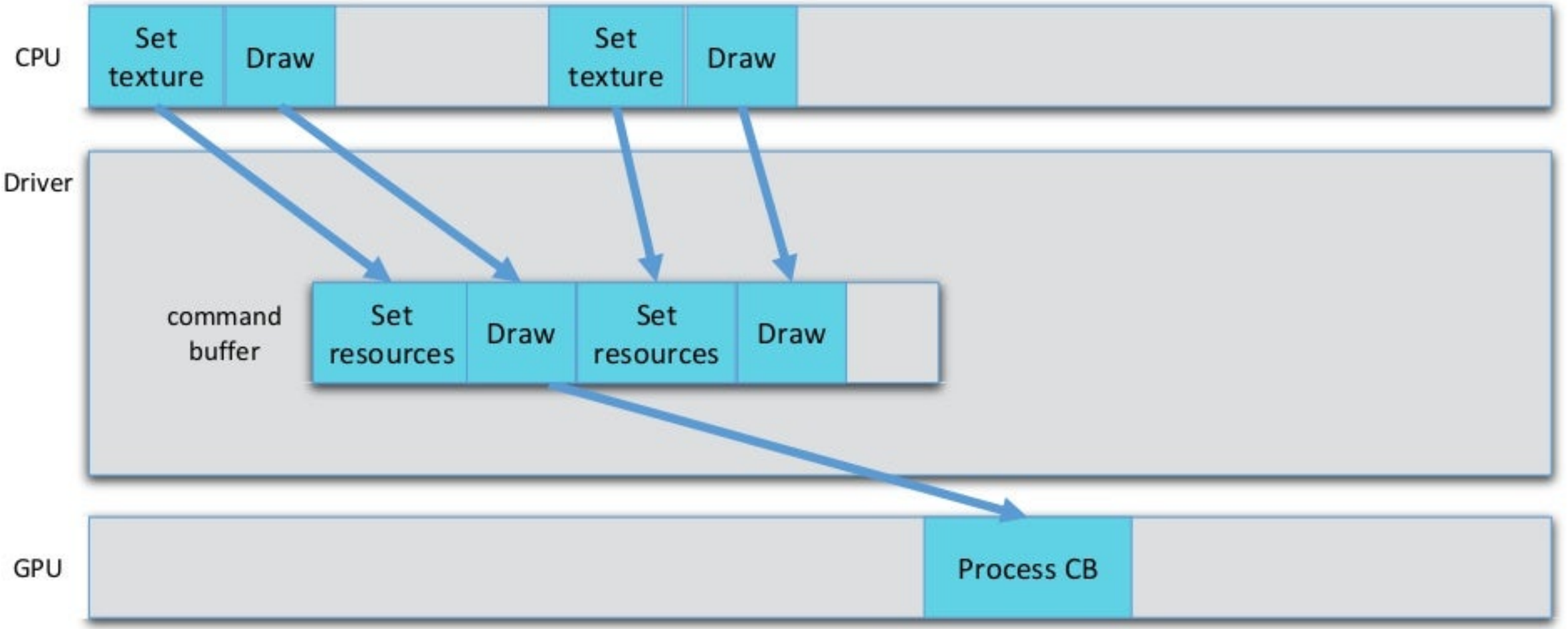
Task-based parallelism: submission to GPU занимает много времени



Task-based parallelism: submission to GPU требует неизменности данных



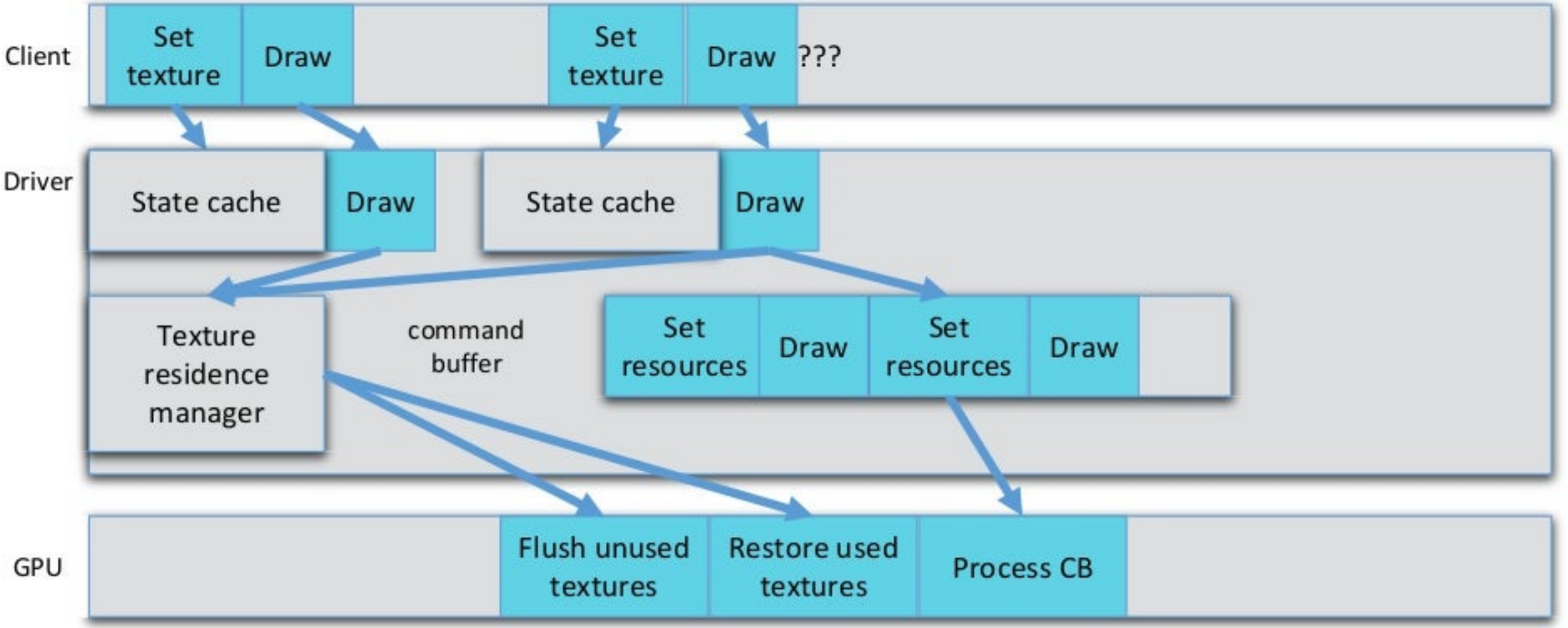
Общение CPU<->GPU



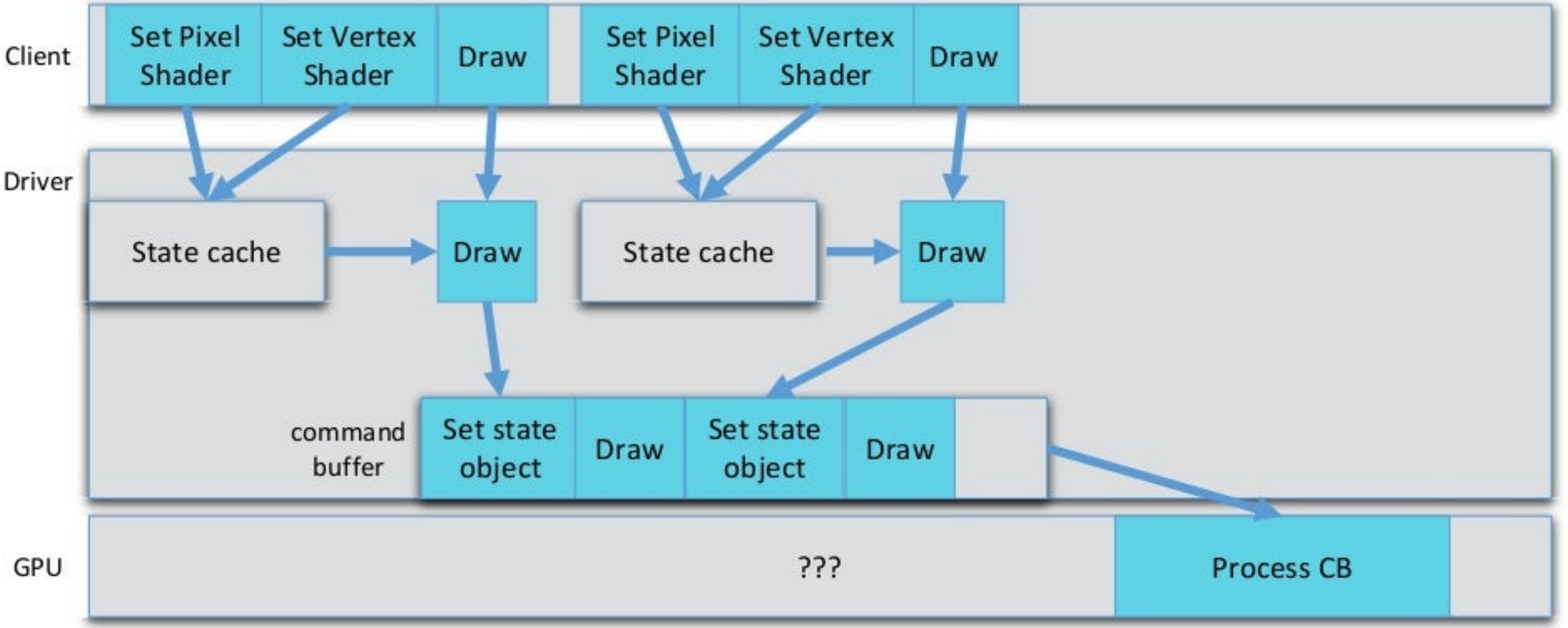
“Старые” графические API

- (Direct3D 11-, OpenGL / OpenGL ES)
- много проверок и дополнительной работы на лету, достаточно дорогие вызовы API
  - захват времени жизни объектов
  - некоторые объекты API далеки от реальных объектов GPU

“Старые” графические API управление резидентностью ресурсов



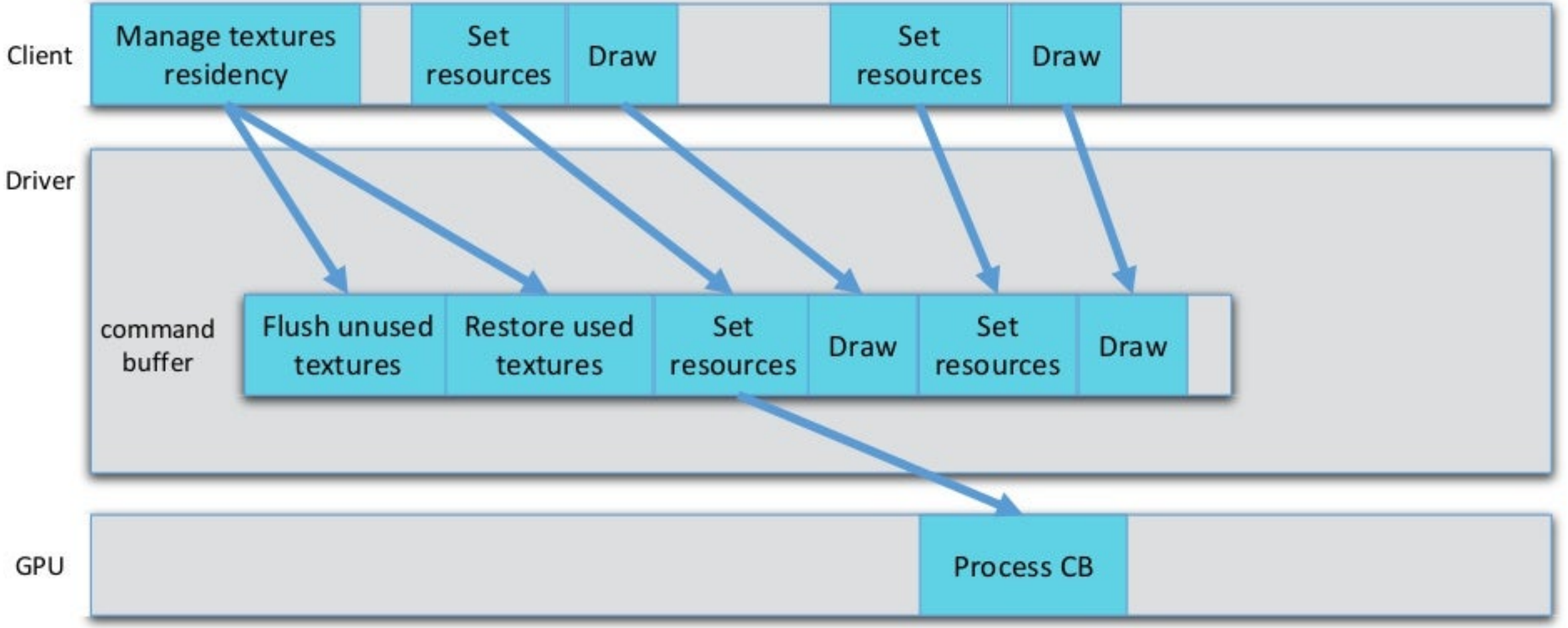
“Старые” графические API управление состоянием GPU



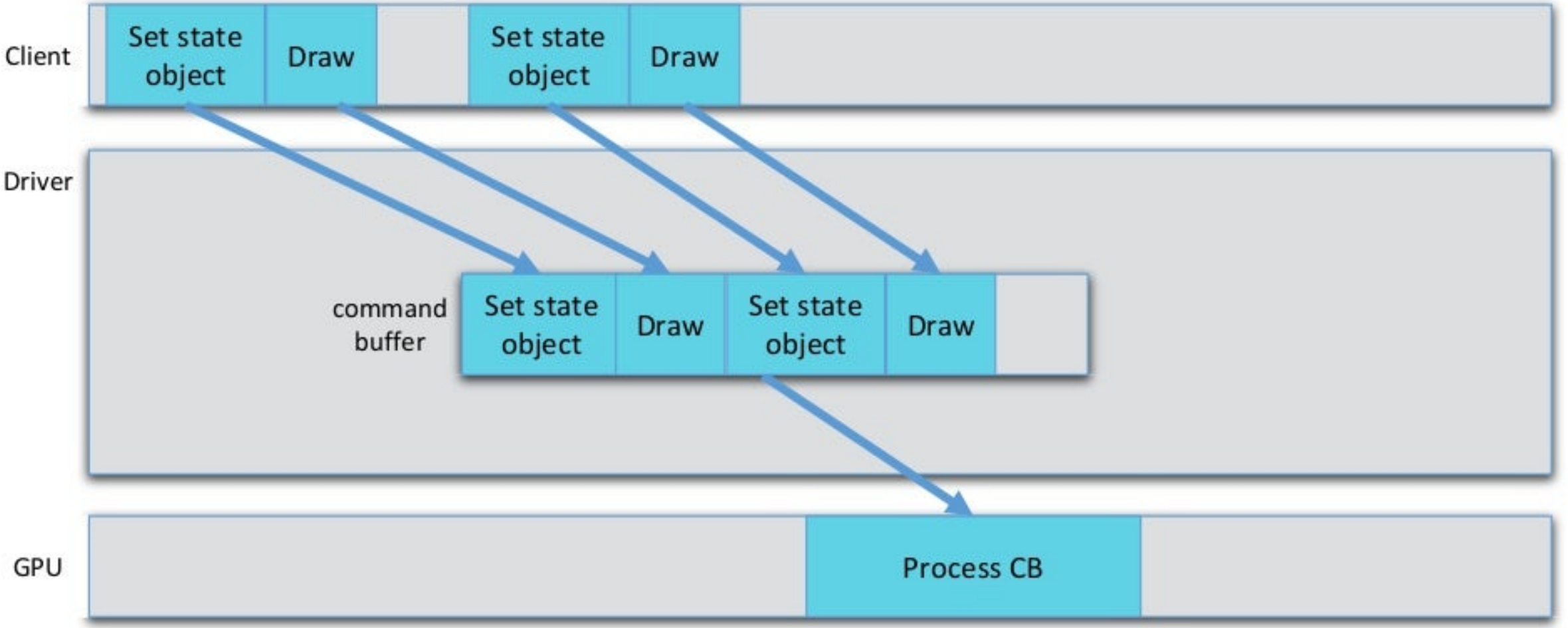
DirectX 12 / Vulkan / Metal

- DirectX 12 - июль 2015, Vulkan - февраль 2016, Metal - сентябрь 2014
- Большую часть работы переложили на клиентский код

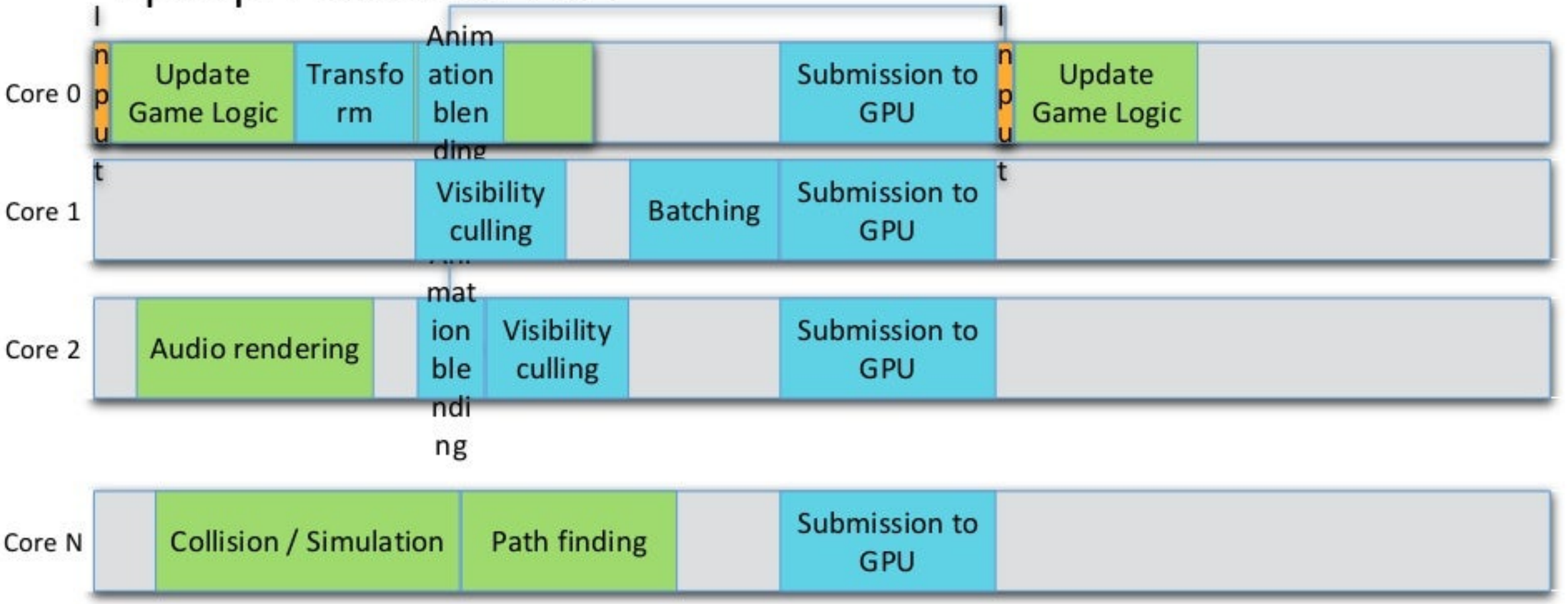
“Новые” графические API управление резидентностью ресурсов



“Новые” графические API управление состоянием GPU



Task-based parallelism + современные графические API



Q&A

- Вопросы?

Links

- https://github.com/dougbinks/enkiTS
- https://www.enkisoftware.com/devlogpost-20150822-1-Implementing_a_lightweight_task_scheduler
- https://www.enkisoftware.com/devlogpost-20150905-1-Internals_of_a_lightweight_task_scheduler
- https://www.slideshare.net/jrouwe/killzone-shadow-fall-threading-the-entity-update-on-ps4
- https://www.gdcvault.com/play/1022164/Multithreading-the-Entire-Destiny
- http://s09.idav.ucdavis.edu/talks/05-JP_id_Tech_5_Challenges.pdf