

Производительность и параллелизм в Spark

МОДУЛЬ 7

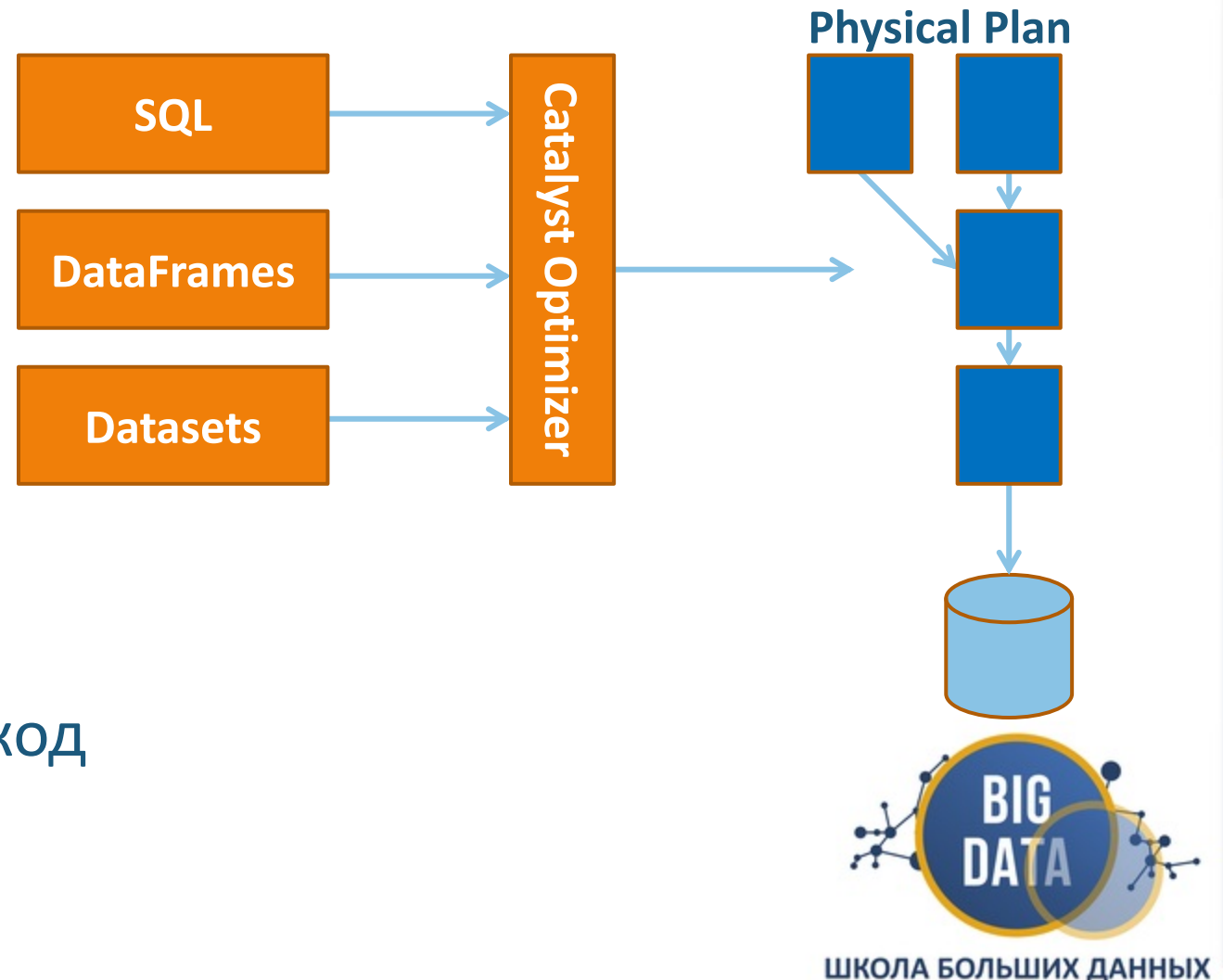


Планы выполнения

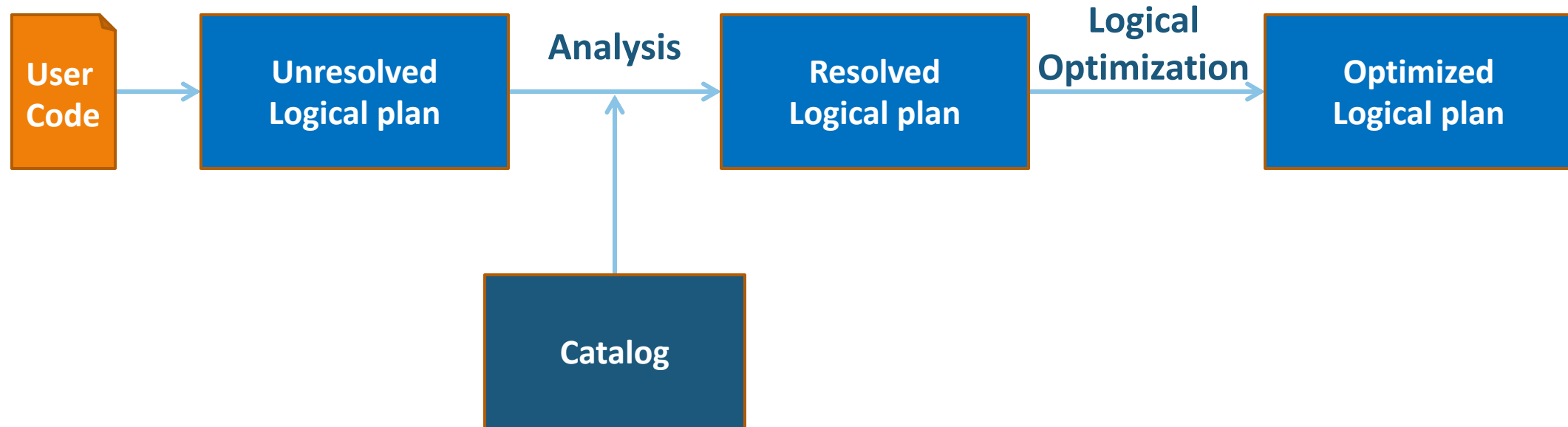


Исполнение spark приложения

- ❖ **код** передается в spark (интерактивно или через “spark-submit”)
- ❖ строится **логический план** исполнения
- ❖ производится **оптимизация**
- ❖ **оптимизированный** (другой) код выполняется на кластере



Логический план исполнения

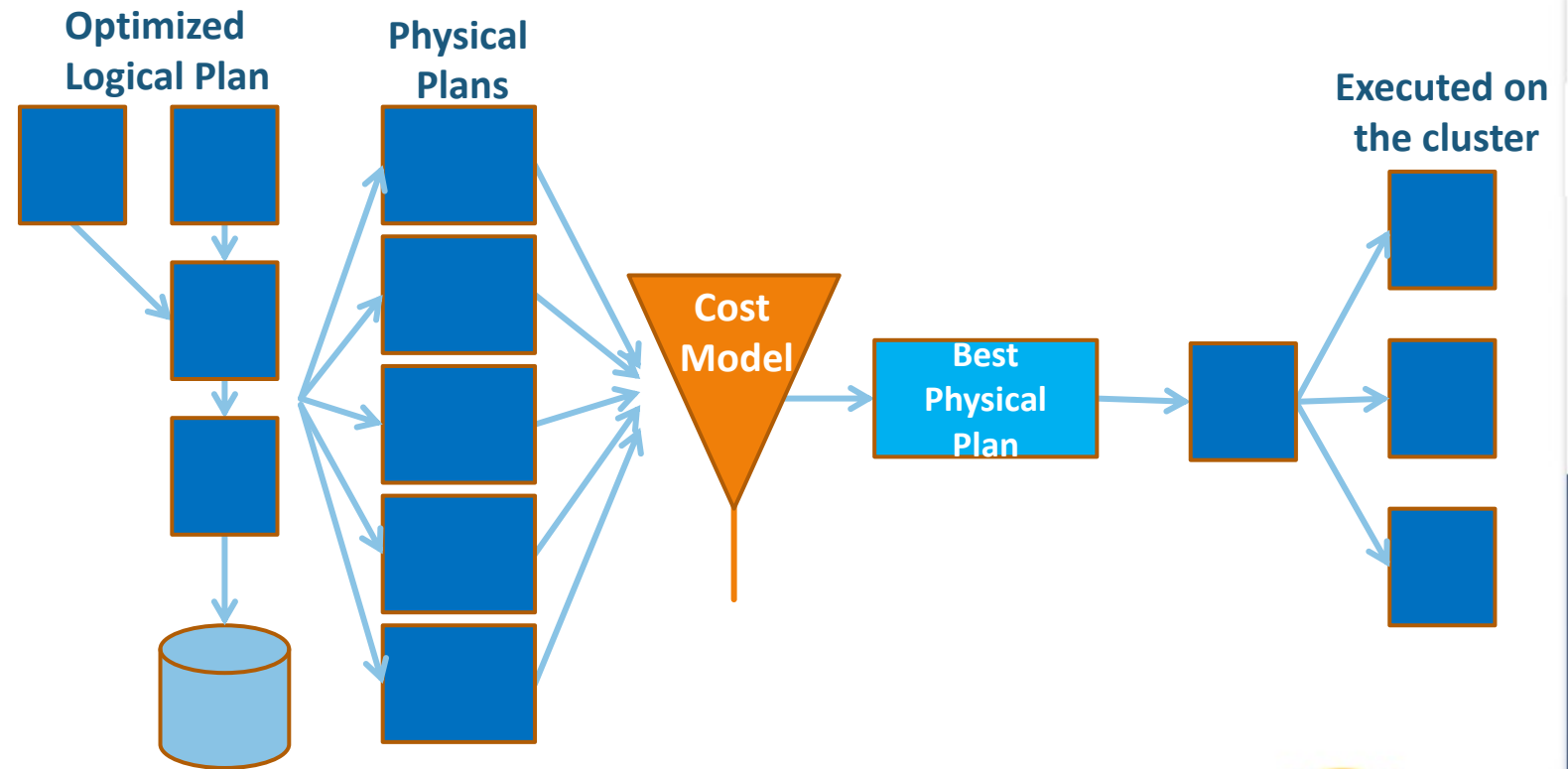


- ✓ сначала строится **“unresolved”** план
- ✓ анализатор производит **привязку** к таблицам и полям
- ✓ **оптимизатор** пытается построить более оптимальный логический план (манипуляции с предикатами и фильтрами)



Физический план исполнения

- ❖ оптимальный логический план транслируется в несколько физических планов
 - ✓ физический план - трансформации RDD
- ❖ выбирается оптимальный
 - ✓ используются модели "стоимости исполнения"



Виды трансформаций

- ❖ **“narrow”** трансформация
 - ✓ фильтры, “map”, ... - действия над одной партицией
- ❖ **“wide”** трансформации
 - ✓ агрегации, сортировка, ... - действия над данными нескольких разделов
- ❖ “wide” трансформации требуют записи на диск (**“shuffle”**)
- ❖ “narrow” трансформации вытягиваются в конвейер (**“pipeline”**) и выполняются без использования диска



Анатомия приложения - job, stage, task

- ❖ **“job”** - как правило одно действие (**“action”**)
- ❖ **“stage”** - трансформации между **“shuffle”**
 - ✓ могут быть пропущены (**“skipped”**), если данные не изменялись
- ❖ **“task”** - действия внутри одной **“stage”**
 - ✓ набор блоков данных и трансформаций



Параллелизм исполнения приложения

- ❖ “job” и “stage” выполняются последовательно
- ❖ “task” могут выполняться параллельно, но
 - ✓ не больше количества **партиций**
 - ✓ не больше количества **эксекьюторов**

Вывод: обеспечиваем достаточное количество партиций и эксекьюторов



Spark performance tuning



Speculative Execution

Механизм **spark** (отключаемый)

- ❖ мониторит задачи ("**task**")
- ❖ **перезапускает** медленный task на другом узле
- ❖ убивает медленный task, использует результаты быстрого

ВАЖНО будет только хуже, если для "медленности" есть причины (см. ниже)



Общие рекомендации

- ❖ Monitor and inspect Jobs
- ❖ Уровень параллелизма - **2-3 tasks per CPU core**
- ❖ Сокращайте размеры обрабатываемых данных
 - ✓ тоже за счет уровня параллелизма
- ❖ Give spark multiple disks for intermediate persistence
- ❖ Higher level APIs are better
- ❖ Avoid collecting large dataframes/RDDs
- ❖ Cache after hard work
- ❖ Always cache after repartition
- ❖ Coalesce or repartition
 - ✓ избегайте огромных партиций - небольшие партиции работают лучше



Дополнительные рекомендации

Improve Shuffle Performance

- ✓ используйте LZF или Snappy Compression (для shuffle)
- ✓ используйте Kryo Serialization

для любителей RDD

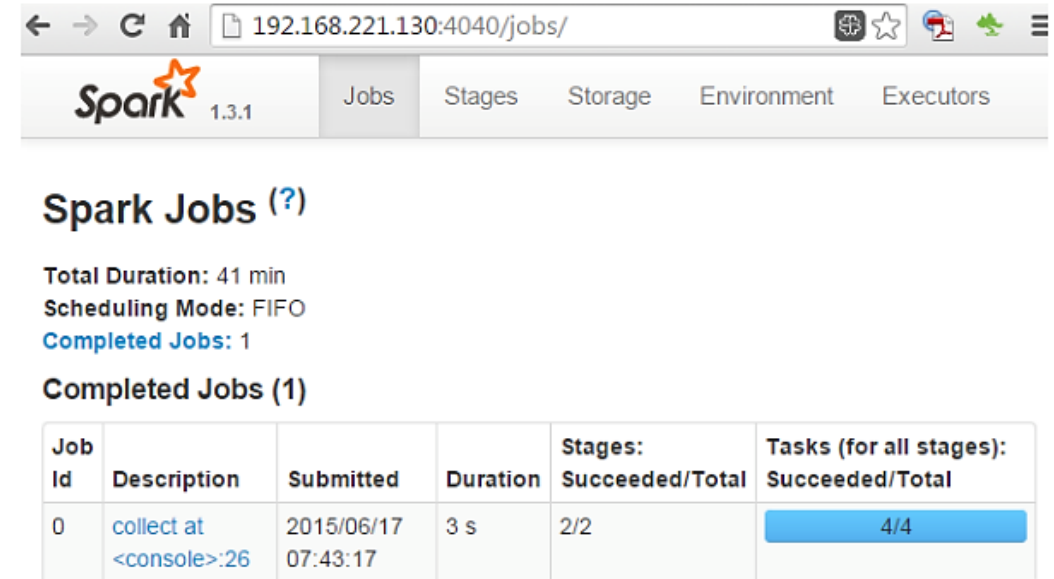
- ✓ Avoid group By Key for associative operations
- ✓ Scala > Java >> Python > R
- ✓ Filter First, Shuffle next
- ✓ Use cogroup
- ✓ A Map after partition By will lose the partition information



Spark WEB UI

- ❖ как правило порт 4040
 - ✓ +1 при занятости (40416 4042, ...)
- ❖ показывает **статус**
- ❖ содержит данные **о метриках**
- ❖ информация об использовании и статусе **кэша**
- ❖ загруженность executor-ов

(посмотрим на практике)



The screenshot shows the Spark Web UI interface. The browser address bar displays the URL `192.168.221.130:4040/jobs/`. The interface includes a navigation bar with tabs for `Jobs`, `Stages`, `Storage`, `Environment`, and `Executors`. The `Jobs` tab is active, showing the `Spark Jobs` section. Below this, summary statistics are provided: `Total Duration: 41 min`, `Scheduling Mode: FIFO`, and `Completed Jobs: 1`. A table titled `Completed Jobs (1)` lists the details of the completed job.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <console>:26	2015/06/17 07:43:17	3 s	2/2	4/4



Вопросы?

