Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

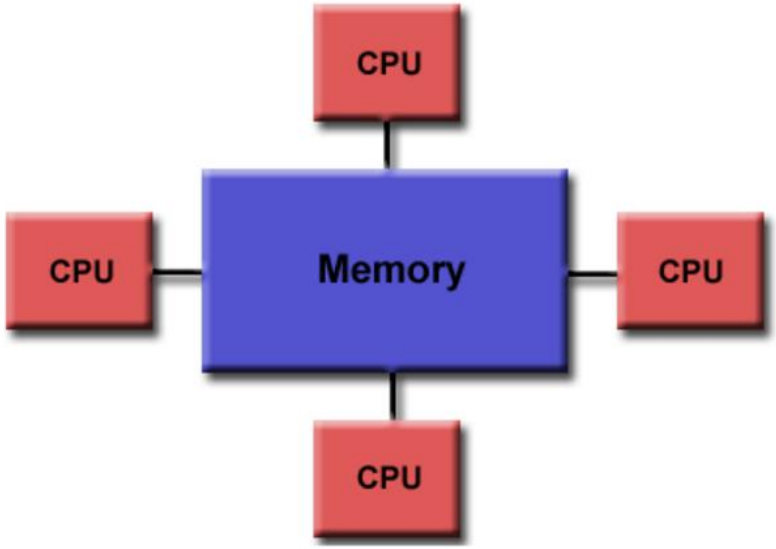
Введение в технологию OpenMP

Эдуард Храмченков

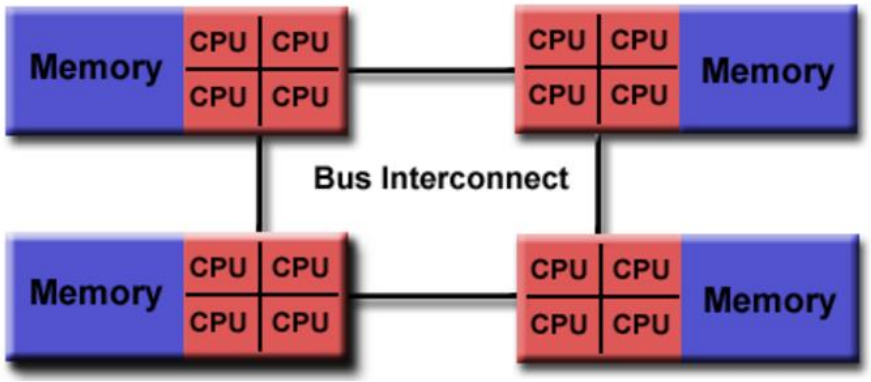
OpenMP

- Это API для разработки многопоточных приложений под SMP архитектуру
- Состоит из:
  - директив компилятора
  - библиотек среды выполнения
  - переменных окружения
- Легкий и удобный способ создавать параллельные программы

OpenMP



Uniform Memory Access



Non-Uniform Memory Access

Казанский федеральный
УНИВЕРСИТЕТ

OpenMP

- OpenMP программы основаны исключительно на использовании потоков
- Поток– это минимальная единица выполнения, которую может выделить ОС
- Потоки существуют в контексте процесса
- Как правило количество потоков совпадает с количеством процессоров/ядер

Fork-join модель

- В начале исполнения существует только мастер-поток, исполняющий последовательные участки кода
- В указанных местах мастер-поток порождает дополнительные потоки (fork)
- Когда параллельный регион заканчивается, управление передается мастер-потоку, другие потоки исчезают (join)

Fork-join модель

Модель памяти OpenMP

- По умолчанию все данные являются общими для всех потоков
- Можно явно указать, что некая переменная является private и у каждого потока своя копия
- Данные каждого потока хранятся в специальном стеке в памяти – ограничен по размеру!

Директивы OpenMP

- #pragma omp directive_name [clause[[,] clause]…]
- Создают группы потоков для параллельного выполнения участка кода
- Определяют как разделять работу среди потоков
- Указывают private и shared переменные
- Управляют синхронизацией потоков

Директива parallel

- #pragma omp parallel [clause[[,] clause]…]
- Указывает на то, что следующий за директивой блок кода будет выполняться параллельно
- Каждому потоку присваивается порядковый номер – у мастер-потока №0
- omp_get_thread_num() – получить номер потока

Директива parallel

- Спецификатор [if(условие)] – запуск в параллельном режиме по условию
- Спецификатор [num_threads(integer)] – позволяет задать количество потоков
- Throw внутри параллельного региона должен не прерывать хода выполнения блока, и должен обрабатываться тем потоком, который его выбросил

Пример 1

- std::cout – разделяемый ресурс, при доступе к нему начинается гонка потоков
- Можно использовать printf() – функция работает быстрее, при небольшом количестве потоков не происходит склеивания строк

Казанский федеральный
УНИВЕРСИТЕТ

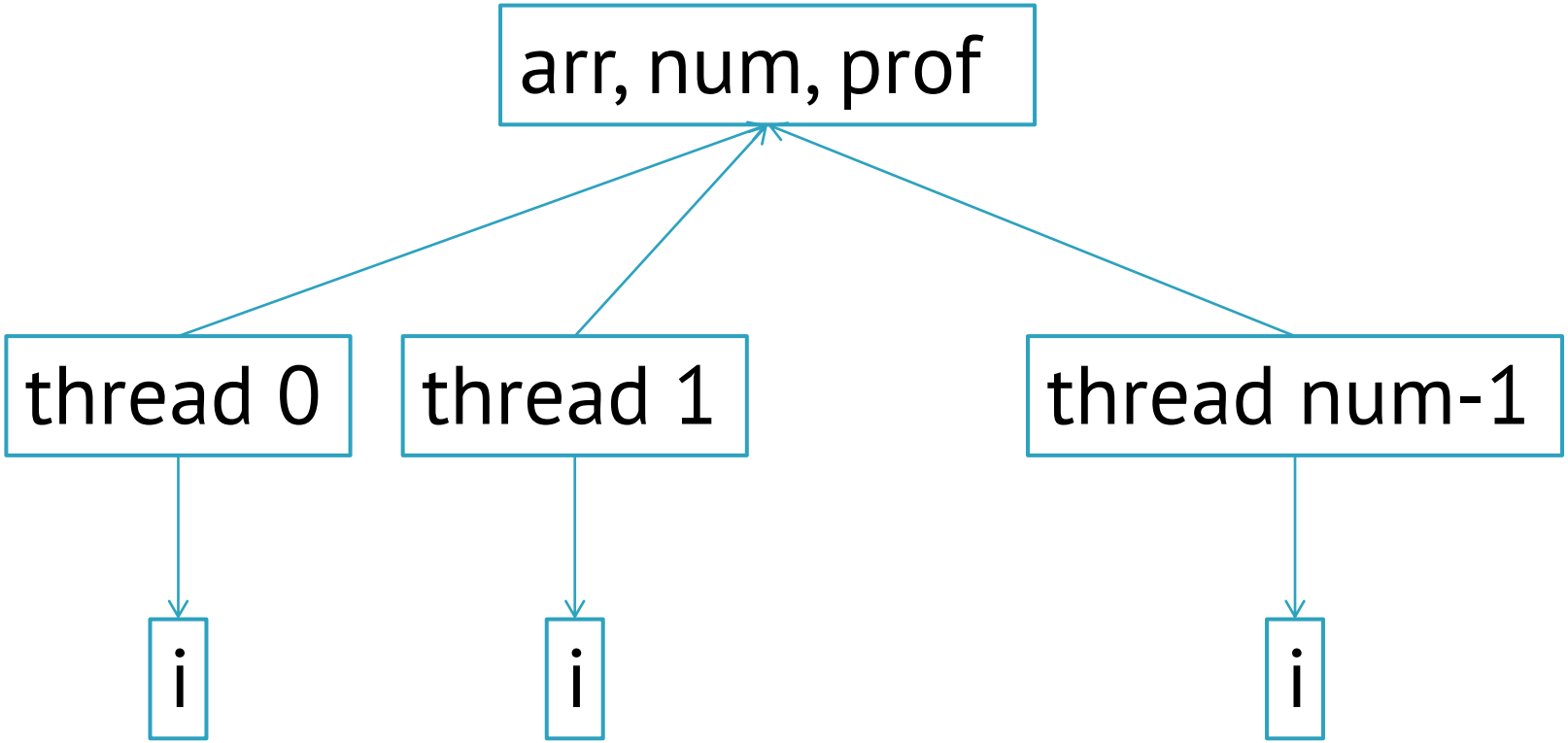
Директива for

- #pragma omp for [clause[[,] clause]…]
- Запускает цикл следующий непосредственно после директивы в параллельном режиме
- Цикл должен иметь каноническую форму for(init-expr; test-expr; incr-expr)
- Возможен краткий вариант записи 2 директив: #pragma omp parallel for [clause[[,] clause]…]

Директива for

- В цикле не должно встречаться goto, exit(), break, return
- Каждый поток имеет свой контекст исполнения – адресное пространство со всеми необходимыми переменными
- Счетчик цикла по умолчанию private

Пример 2



Параллелизация циклов

- В начале блока цикла происходит fork потоков, в конце цикла – барьер синхронизации и join потоков
- В случае, если параллелизуется внутренний цикл, возникают большие издержки на fork/join
- Если параллелизуется внешний цикл, то fork/join только один раз

Параллелизация циклов

- Если цикл содержит вложенные циклы, то в большинстве случаев следует параллелизировать внешний цикл
- Степень дробления – количество вычислений между синронизациями/коммуникациями
- Чем больше степень дробления, тем выше производительность параллельного кода

Казанский федеральный
УНИВЕРСИТЕТ

Пример 3

- Вложенный цикл
- Проблема: у каждого потока своя private переменная i, а j – shared для всех потоков
- Решение – явно объявить переменную j как private
- Спецификатор private(список переменных)
- Каждый поток получает свою копию этой переменной для своего контекста

Пример 4

- Результат хранящийся в переменной a зависит от порядка выполнения тела цикла потоками
- Если объявить ее как private – значение на выходе будет одним и тем же
- Значение private переменных не определено на входе и выходе из блока
- По стандарту начальное значение такой переменной не гарантируется

Пример 4

- Спецификатор firstprivate – private переменная в начале блока инициализируется значением этой переменной, заданным ранее
- Спецификатор lastprivate – значение private переменной после завершения блока будет содержать значение полученное в ходе последней «логической» итерации

Спецификатор nowait

- Указывает потоку не ждать остальных в конце блока, а приступать к выполнению следующего блока, если он есть

```
#pragma omp parallel
#pragma omp for nowait
for{…}
```

- В конце параллельного региона барьер есть в любом случае

Казанский федеральный
УНИВЕРСИТЕТ

Спецификатор schedule

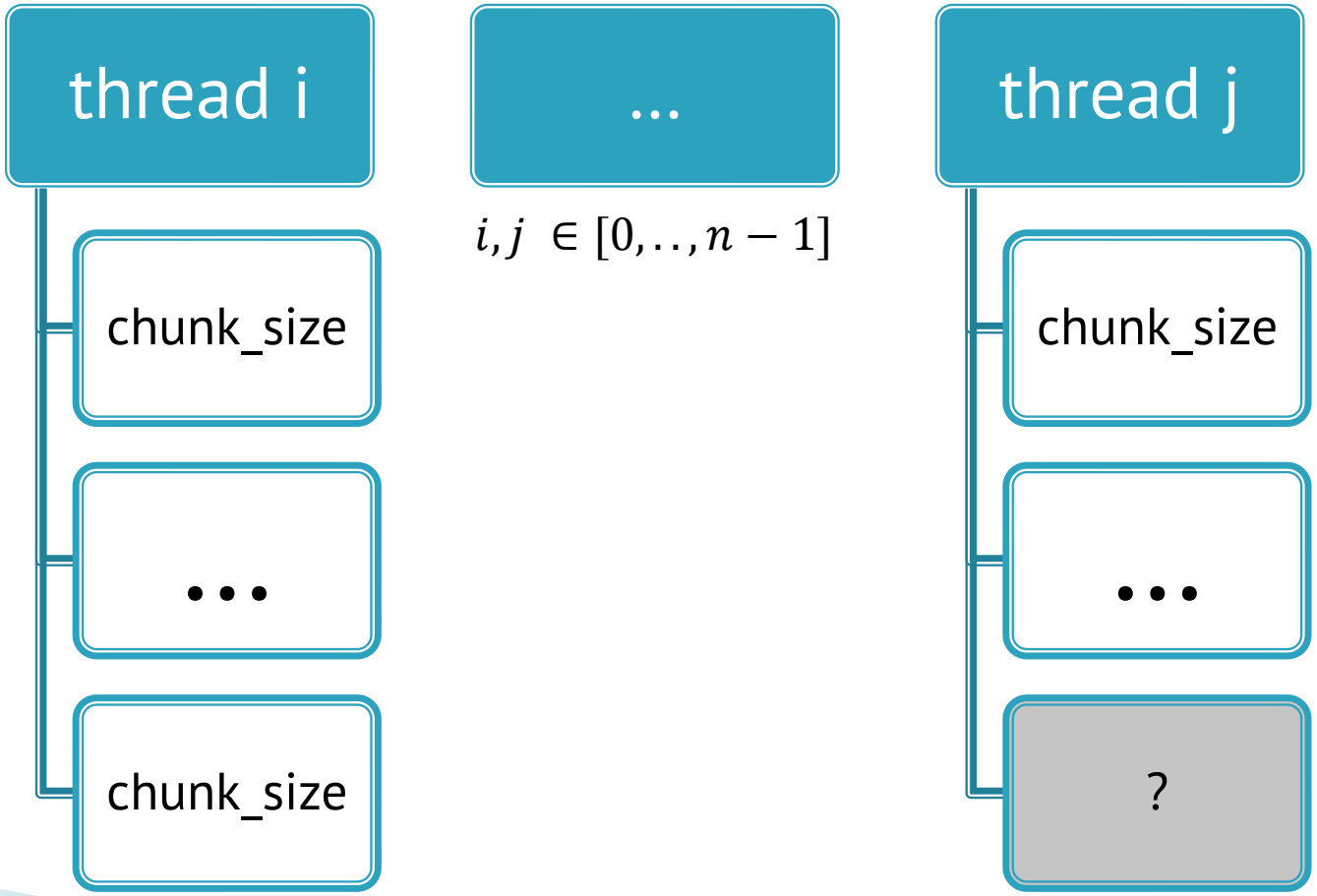
- Поддерживается только для циклов
- schedule(kind[, chunk_size])
- Управляет распределением итераций цикла между потоками
- Параметр chunk_size не обязан быть константным выражением
- Существуют 4 типа распределения, которые задаются параметром kind

schedule static

- static – наиболее простой, по умолчанию используется в тех случаях, когда нет явного задания schedule
- Итерации делятся по потокам блоками в соответствии с размером chunk_size
- Последний блок может иметь меньший размер
- Если chunk_size не задан итерации делятся примерно поровну

Казанский федеральный
УНИВЕРСИТЕТ

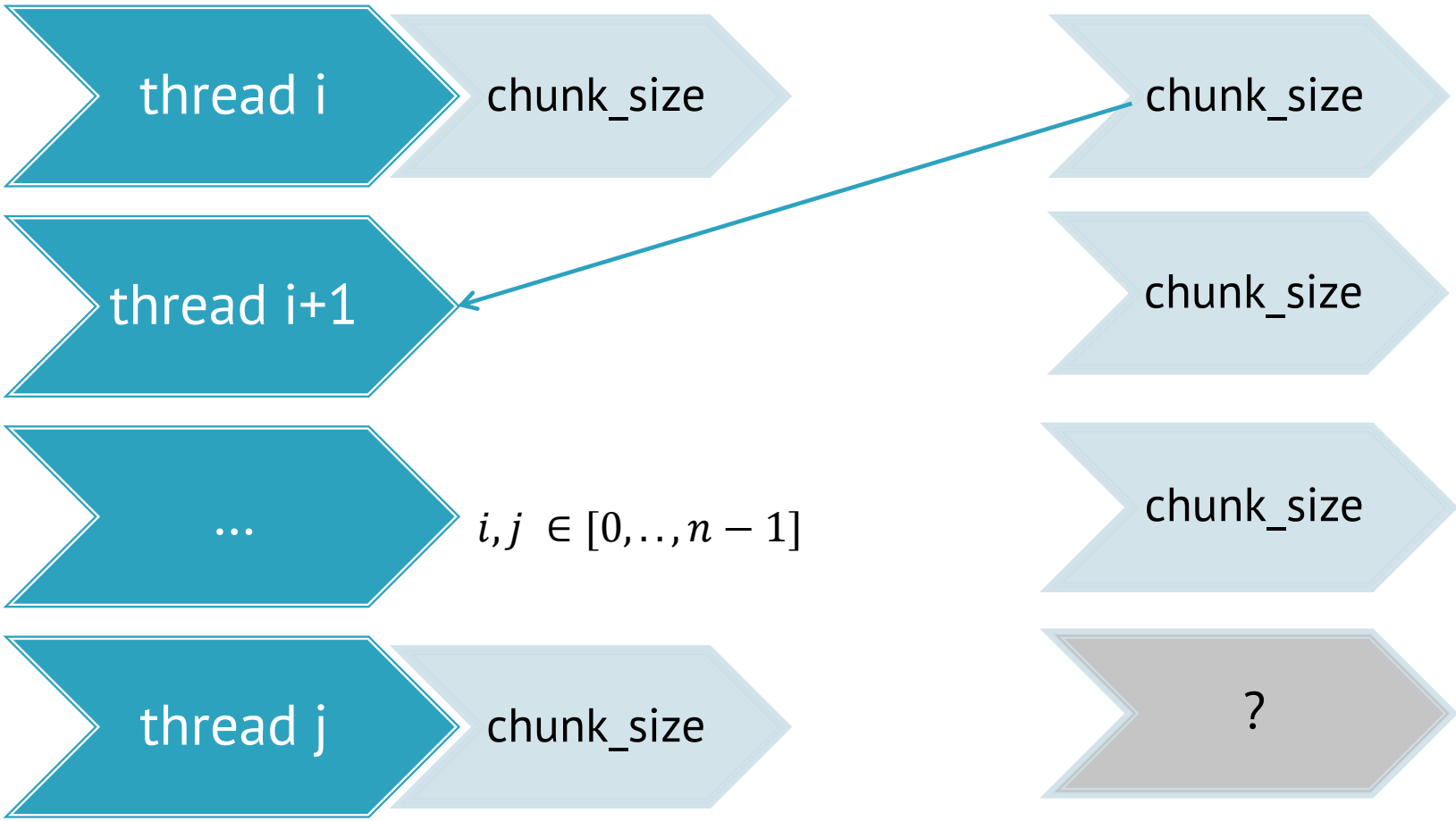
schedule static



schedule dynamic

- Используется для плохо балансированных и непредсказуемых нагрузок в цикле
- Итерации назначаются потокам блоками размером chunk_size
- Поток исполняет блок, затем запрашивает следующий, и т.д. пока не закончатся блоки
- Размер chunk_size по умолчанию = 1

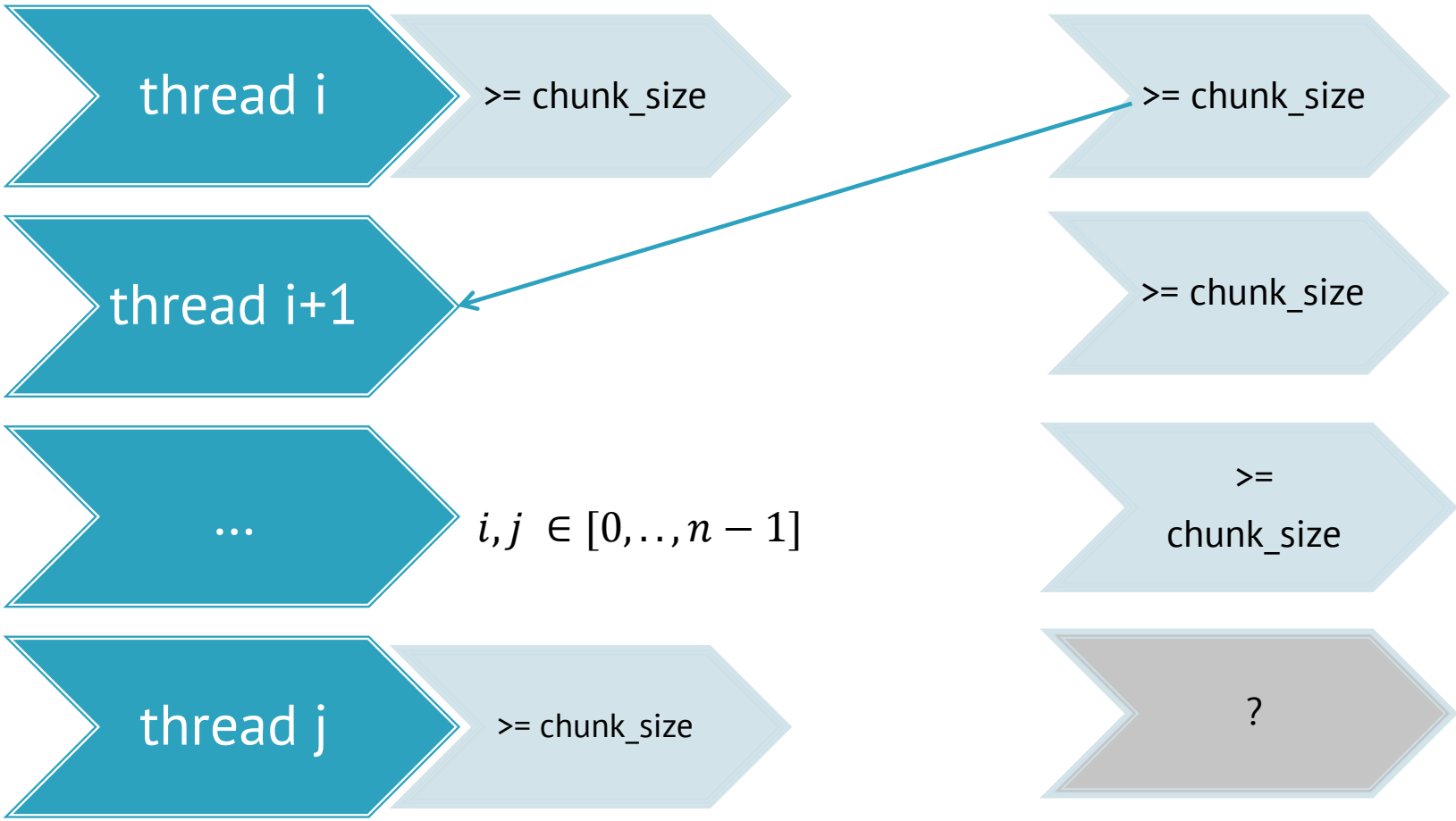
schedule dynamic



schedule guided

- По схеме схож с dynamic – поток выполняет блок, потом запрашивает следующий, пока не кончатся блоки
- Разница в размере блоков – в данном случае параметр chunk_size определяет *минимальное* количество итераций в блоке
- Реальное количество итераций в блоке определяется компилятором

schedule guided



schedule runtime

- Выбор между тремя типами распределения нагрузки происходит во время выполнения кода
- Выбор определяется значением переменной окружения OMP_SCHEDULE
- OMP_SCHEDULE является строкой, которая подставляется в спецификатор schedule

Казанский федеральный
УНИВЕРСИТЕТ

Пример 5

- Используется тип runtime для спецификатора schedule
- Для установки переменной OMP_SCHEDULE используется функция omp_set_schedule(type, chunk_size)
- type – строковая переменная вида omp_sched_static/dynamic/guided

Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru