Лекция 14

Алгебраическое программирование и инсерционное моделирование

Параллельное функциональное программирование

Вызовы и развертывания

$$f_i(x_1, x_2, \ldots) = E_i(f, x),$$
$$i = 1, \ldots, n, x = (x_1, x_2, \ldots)$$

$f_i(t_1, t_2, \ldots)$	вызов
$E_i[x_1 := t_1, x_2 := t_2, \ldots]$	развертывание

Вхождение вызова в *E*, отличное от *E* – **внутренний вызов**

Операционная семантика
параллельного функционального программирования

$$\frac{t = G[x_1 := G_1, x_2 := G_2, \ldots]}{t \to simpl(G[x_1 := G_1', x_2 := G_2', \ldots])}$$

простое параллельное развертывание

$$\frac{t = G[x_1 := G_1, x_2 := G_2, \ldots], G_1 \to F_1, G_2 \to F_2, \ldots}{t \to simpl(G[x_1 := F_1', x_2 := F_2', \ldots])}$$

рекурсивное развертывание

E – вызов, E' – развертывание, G_i – вызовы, $G_i \to F_i$ – простое параллельное или рекурсивное развертывание (рекурсивно)

простое развертывание $G \xrightarrow[0]{} F$

рекурсивное развертывание $G \xrightarrow[n]{} F$

$$G_1 \xrightarrow[n_1]{} F_1, G_2 \xrightarrow[n_2]{} F_2, \ldots, n_1, n_2, \ldots \le n$$

$$G[x_1 := G_1, x_2 := G_2, \ldots] \xrightarrow[n+1]{} simpl(G[x_1 := F_1, x_2 := F_2', \ldots])$$

Еще одно простое развертывание

G_i внутренние вызовы

Выполнение одного перехода:

1. **Выделить и частично упорядочить некоторое множество вызовов по уровням;**
2. **Развернуть вызовы нижнего уровня;**
3. **Пока есть неразвернутые вызовы, развернуть вызовы следующего уровня.**

Пункты 1-3 можно выполнять асинхронно и параллельно, сохраняя последовательное выполнение по частичному порядку.

$$\frac{t = G[x_1 := G_1, x_2 := G_2, \ldots]}{t \to simpl(G[x_1 := G_1', x_2 := G_2', \ldots])}$$

$$\frac{t = G[x_1 := G_1, x_2 := G_2, \ldots], G_1 \to F_1, G_2 \to F_2, \ldots}{t \to simpl(G[x_1 := F_1', x_2 := F_2', \ldots])}$$

Управление параллельным вычислением функциональных выражений по определяемым функциональным символам

Определяемые функциональные символы распределяются по процессам (ветви или процессоры) так, что за каждый символ отвечает только один процесс.

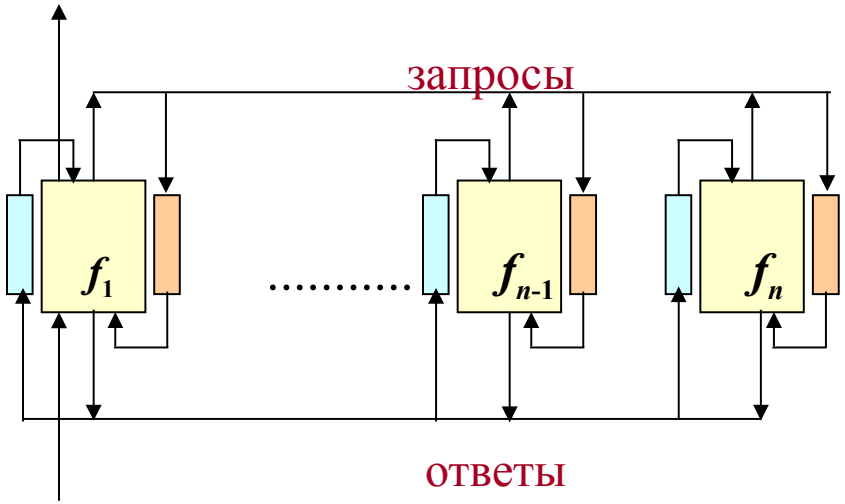
Главный процесс получает задание извне и возвращает результат вовне. Внутренние вызовы раздает соответствующим процессам в виде запросов.

В произвольный момент времени состояние процесса характеризуется очередями запросов и ответов. Если очереди пусты, процесс ожидает. Если не пусты, делает шаг обработки каждого запроса из очередей.

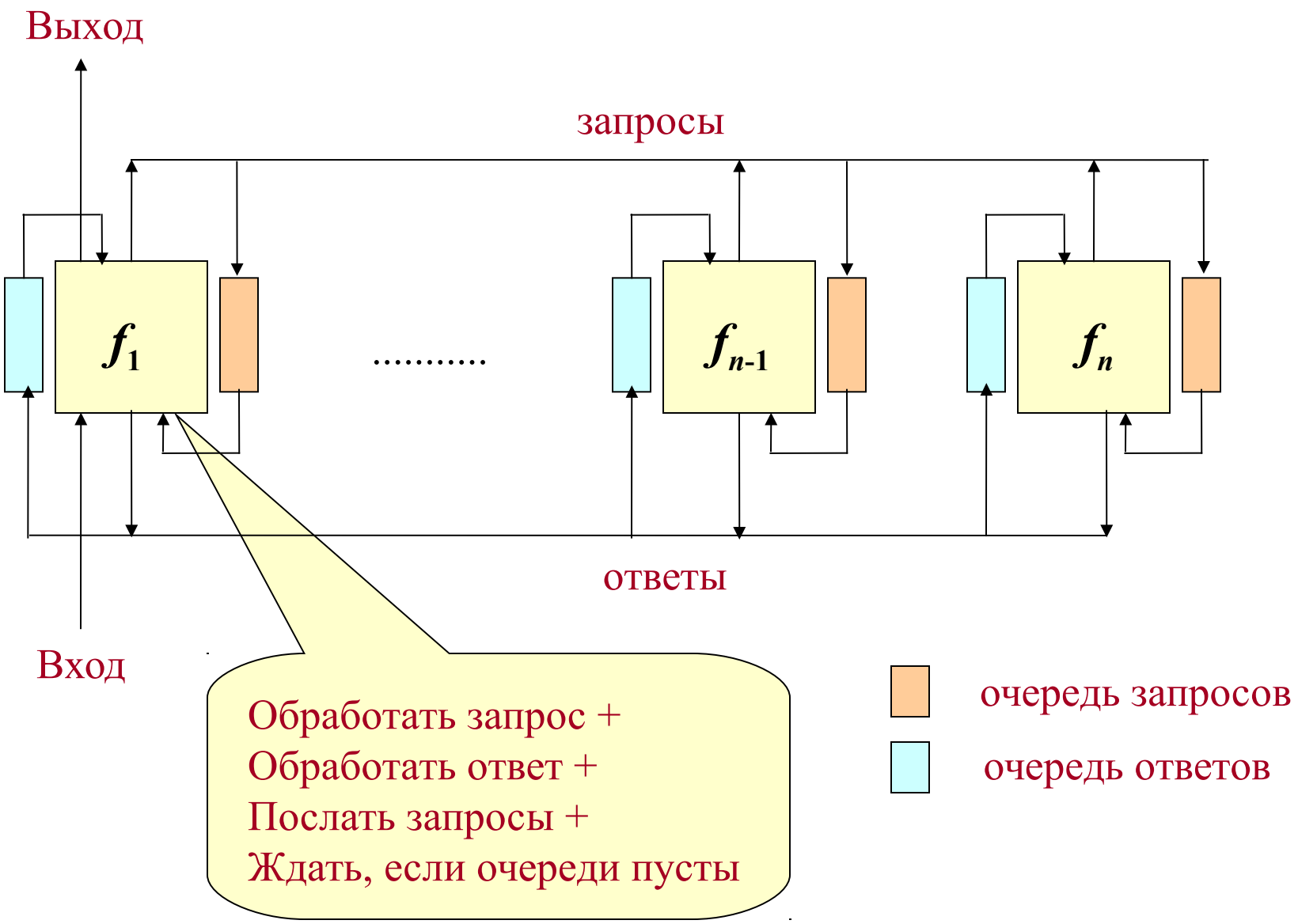
Метод перекрестных запросов. Требуется точная координация запросов и связей между ними (формат запросов и ответов).

Для **нестрогих функций** требуется передача информации о прекращении обработки некоторых запросов.

Хорошо работает в системах с общей памятью.



Строгая функция – требует вычисления всех аргументов
Нестрогая функция: if *x* then *a* else *b*

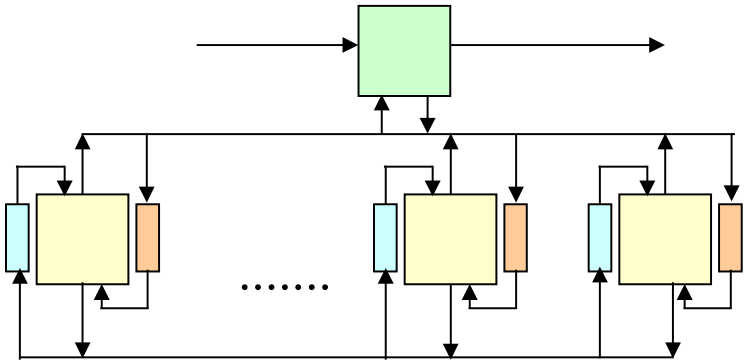

Выход
запросы
f_1
.....
f_{n-1}
f_n
ответы
Вход
Обработать запрос +
Обработать ответ +
Послать запросы +
Ждать, если очереди пусты
очередь запросов
очередь ответов

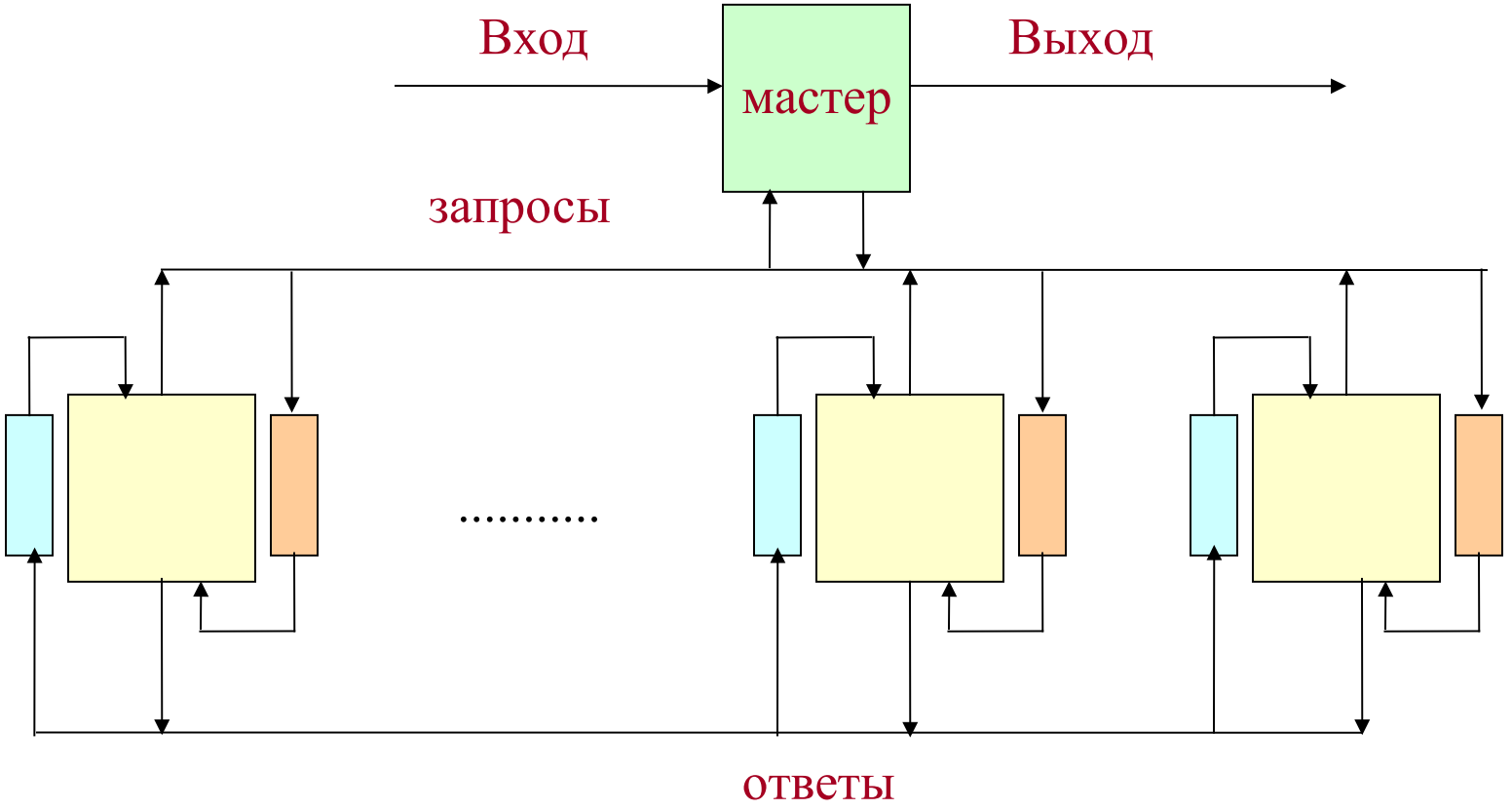
Управление по схеме мастер – работник

Необходимо в распределенных системах. Или в случае, когда возможны перегрузки по вызовам одной и той же функции.

Все запросы передаются через мастера. Он владеет информацией о загрузке каждого процесса и в первую очередь загружает тех, у кого очередь меньше, чем у других.

При высоком росте количества заданий (переполнение памяти) требуется некоторые ветви тормозить, отдавая преимущество другим. Хорошо, если удается прогнозировать.





Генерация процессов динамическая