Kitchen Analogy:

**Flynn’s Taxonomy:**  As in our reading, the author describes that the process can take single or multiple instructions as well as single or multiple data at a time. The same can happen in the restaurant’s kitchen. If our restaurant is small and there is only one person taking an order than it can be seen that the processor is taking only one instruction/order at a time. If there are more than one person to take an order than that is multiple instructions at a time. Same way if the cooking place/stoves are only one set, then it is a single data at a time. If there are multiple stove collection inside a kitchen then it is multiple data/resources.

SISD: 1 Order Receiver(OR) 1 Stove Set(SS)  
SIMD: 1 OR 1+SS  
MISD: 1+ OR 1stove set  
MIMD: 1+ OR 1+SS

**Amdahl’s law:**  
As Amdahl’s laws mentioned there applies the law of diminishing return with parallelizing the processor same happens in the kitchen. We can smoothen and parallelize the task only to a certain extent. In the real world if we try to over parallelize tasks then it actually gets slower after a certain point, in computer system processes will be unused. In either case, we are putting more resources while gaining nothing.

**Parallel Computer Memory Architecture:** In Parallel computing, we have designs called shared memory architecture, distributed memory architecture. In **shared memory architecture** different processes use the same memory. In our analogy, this is obtained by if a 1+ chef uses a stove cluster in only one place; when all the cooking must be done in one and only one place no matter how many chefs are cooking food.   
 While in the **Distributed** **system** each process has its own memory (each chef has their own stoves). In computing the processor talks to each other with the help of the I/0 interface and interconnection network. Here in our analogy, this operation is achieved by each chef assigning their task and with a voice command. Here, I/0 interface is a mouth and ear while the order comes straight come straight to the kitchen and since everybody knows what their task is the process will be streamlined.

Parallel Programming Models: We talked most of the things about **shared Memory Model(without threads) and distributed memory/message-passing model** above here we will discuss threads model, data-parallel model, SPMD and MPMD.

**Thread model** is similar to the message-passing model, but in this case, we don’t have a sender and receiver. A process can have multiple threads and there can be another process executing a tread from some another process. This process can be understood in the following scenario. Let’s say there are 3 chefs in the kitchen. A kitchen gets 3 orders, one chef starts two orders because they have a similar process and another process takes the remaining order. In this case, our 3rd chef can wait for an order. Instead, he starts executing the small task to help both chefs like making all the ingredients ready and taking food out of the freezer etc.

Another is the **data**-**parallel** **model** where two processes can divide the task and execute concurrently. Like multiplying two independent arrays. In our analogy, it can be explained as one chef is making fried rice and another is making noodles. But it has a disadvantage too. Think if they ran out of boiled noodles and they have to boil a new packet. Now the chef has to wait until the noodles are boiled. He is dependent on the result of someone else.

In **SPMD** a copy of a program is given to every task, but each task only executes certain parts of the job. It is difficult to explain only inside the kitchen. Let's say one order comes and the customer wants it but the chef who is specialized in that is not there so they have to order from another restaurant and give to their customer without informing them. Here order comes to 2nd chef he transfers to manager and manager brings food from another restaurant. 2nd chef redecorates and serves the customer. So, the order/program was one but there were many tasks. This model can be message passing or shared memory or both/hybrid.

An **MPND** model can be very chaotic and hard to control in a kitchen. It is a case where the order comes, and one chef does a small part of one order and the other do other smaller part of different orders. It can be done in a restaurant but it is not practical.

**Partition Strategy:**

There are two types of partition strategy, Domain and Functional. In **domain,** every task is performed on part of the data. If we have four burners in one stove set. A chef can heat oil in one, fry vegetables on another before cooking, and many more. It is basically like a domain specialization.   
While in **Functional decomposition,** as far as I can understand is more like pipelining in the computer system. Where lots of instructions are executed in parallel but are fetched one at a time. In functional each task does small pieces of the main task. It is like an assistant chef helping the executive chef.

**Communication Consideration  
Latency Vs Bandwidth:** Here in our scenario let’s say Latency is a time and Bandwidth is the quality of dish we can produce. We can either make a lot of dishes faster or we can take extra time and decorate our food. While the perfect balance gives OS better performance, in our analogy it makes our restaurant profitable.

**Cost**: While having more hardware running parallelly can faster the process but if it can be achieved with better quality and less hardware than it can be more efficient. Same in the kitchen case is it better to have more employees with average experience or fewer employees with high skills.

Synchronous Vs Asynchronous: We all know what it means synchronous and asynchronous execution of an instruction in a computer system. In kitchen, the order receives the order and he doesn’t care when the order will be sent to the customer and keep doing his job; it is asynchronous. While the chef cannot start the order until the order receiver takes an order so the chef becomes synchronous in this case.

Stove Collection: Memory

Chefs: Processers

Order Receiver: I/0

Stove Set: data