algorithm, allowing React to better prioritize and schedule updates, handle asynchronous rendering, and support new features like Suspense and concurrent rendering.

Here are some key points about React Fiber:

1. **Incremental Rendering**: React Fiber introduces the concept of incremental rendering, which means that the rendering work can be split into smaller chunks or "fibers." This allows React to prioritize and schedule updates more efficiently, leading to smoother and more responsive user interfaces.

2.**Scheduling and Prioritization**: Fiber introduces a priority-based scheduling algorithm, enabling React to prioritize updates based on their urgency. This helps ensure that high-priority updates, such as user interactions or animations, are processed without delay, while lower-priority updates are deferred as needed.

3. **Concurrent Mode**: React Fiber introduces Concurrent Mode, which allows React to work on multiple tasks concurrently, including rendering, layout, and data fetching. This enables React to better utilize available resources and provide a more seamless user experience, even under heavy load.

4. **Error Handling and Interruptibility**: Fiber improves error handling and interruptibility in React applications. It allows React to recover gracefully from errors without crashing the entire application and provides mechanisms for handling asynchronous tasks more effectively.

5. **Support for Suspense**: React Fiber introduces support for Suspense, a declarative way to manage asynchronous operations, such as data fetching or code splitting. Suspense allows components to suspend rendering while waiting for data to load, improving the perceived performance of React applications.

6. **Backwards Compatibility**: Despite being a major internal rewrite, React Fiber maintains backwards compatibility with existing React applications and APIs. This allows developers to gradually adopt Fiber without having to rewrite their entire codebase.

Overall, React Fiber represents a significant advancement in the React ecosystem, enabling React to better handle complex user interfaces, asynchronous operations, and performance optimization. It lays the foundation for future improvements and innovations in React development.

useReducer

- Double Hashing: Uses a secondary hash function to calculate the interval between probe sequences, reducing the likelihood of clustering.
- Open addressing requires careful handling of deletion (e.g., tombstone markers) and resizing to maintain performance.

3. **Robin Hood Hashing**:
- Robin Hood hashing is an extension of linear probing that attempts to reduce clustering by swapping elements to balance probe lengths.
- When inserting a new element, it compares the probe length (the number of slots between the initial and current position) with the probe length of the existing element at that position.
- If the new element has a shorter probe length, it replaces the existing element and continues probing for the displaced element.
- This strategy aims to distribute elements more evenly, reducing the maximum probe length and improving performance.

Each collision resolution strategy has its advantages and disadvantages, and the choice depends on factors such as the expected number of collisions, the size of the hash table, and the characteristics of the keys being stored.