1) **Requirement A: Ticket sales**

Recommendation: An ACID-compliant relational database like PostgreSQL.

Why it's the right choice:

100% reliability: The ACID (Atomicity, Consistency, Isolation, Durability) properties of a relational database guarantee that transactions are processed reliably, even during system failures.

No double-selling: The isolation property ensures that concurrent booking requests for the same seat are handled as if they were processed one after the other. A transaction to book a seat will be completed or completely rolled back, preventing any double-selling mishaps.

Immediate seat availability: The atomicity property ensures that if a booking fails, all partial changes are undone, and the seat becomes immediately available again. The entire operation is treated as a single, "all or nothing" unit.

CAP Theorem:

For ticket sales, **Consistency and Partition tolerance (CP)** are prioritized over Availability. It's acceptable if the system is briefly unavailable, but it must never sell the same seat twice. PostgreSQL ensures strong consistency across transactions, which is critical here.

Trade-off & Biggest Risk:

The trade-off is reduced availability — during heavy load or network partitions, the system may reject bookings rather than risk inconsistency. The biggest risk is downtime during peak sales, which can frustrate users. This can be managed with database clustering and failover strategies.

2) **Requirement B: News feed**

Recommendation: A BASE-based NoSQL database such as a document-oriented like MongoDB.

Why it's the right choice:

Extremely fast and highly available: These databases are built on BASE (Basically Available, Soft state, Eventual consistency) principles, prioritizing availability and performance over immediate consistency. They are designed to stay online and respond quickly even when dealing with immense traffic.

High-volume concurrent operations: NoSQL databases can be horizontally scaled, meaning you can add more servers to handle millions of simultaneous users reading from and posting to the feed without a single point of failure.

Tolerates eventual consistency: For a news feed, a delay of a few seconds for new posts to appear across the city is acceptable. The "eventual consistency" property of BASE systems handles this perfectly by ensuring the data will become fully consistent over time.

CAP Theorem:

For news feed, **Availability and Partition tolerance (AP)** are prioritized over Consistency. The system must always stay online and scale to millions of users. Eventual consistency is acceptable since a post appearing a few seconds late does not harm user experience.

Trade-off & Biggest Risk:

The trade-off is weaker consistency — users may briefly see outdated or missing posts. The biggest risk is synchronization lag under extreme loads, which could impact user experience. This can be managed by tuning replication and using caching layers for faster reads.

3) **Requirement C: User profiles**

Recommendation: A document-based NoSQL database like MongoDB.

Why it's the right choice:

Flexible and evolving structure: Unlike rigid relational databases that require a predefined schema, document databases are schemaless. This means you can easily add new fields for bios, social media links, or photo portfolios to user profiles as the application evolves over time, without complex database migrations.

Ideal for semi-structured data: A document model can store all the profile information for a user in a single "document." This makes it perfect for managing the varied, semi-structured data described in the requirement, such as some users having a bio while others do not.

Agile development: The flexibility of this database type complements an agile development process, allowing developers to quickly iterate and add new profile features based on user needs.

CAP Theorem:

For user profiles, a balance between **Consistency and Partition tolerance (CP)** is often preferred. Users should see their updates immediately and reliably, while still being distributed across servers.

Trade-off & Biggest Risk:

The trade-off is reduced availability — in case of partitions, some profile updates may be delayed to maintain consistency. The biggest risk is profile data conflicts across replicas, which can be mitigated with conflict resolution policies and strong consistency modes for critical fields.