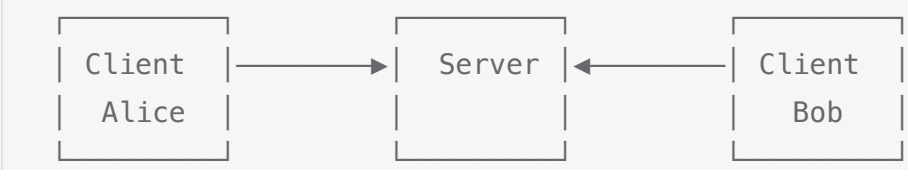
Core Concepts

Understanding these fundamental concepts will help you build powerful P2P applications with Pear.

What is Peer-to-Peer?

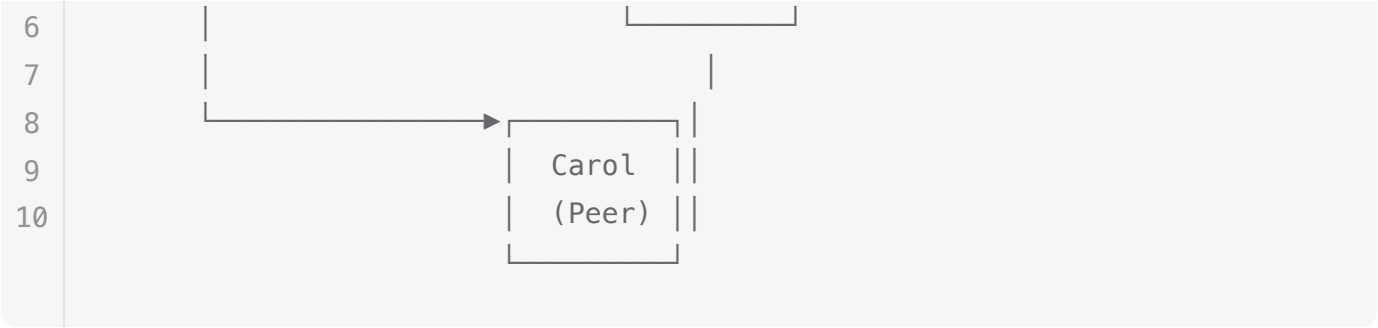
Traditional Architecture (Client-Server)



Characteristics:

- All data flows through a central server
- Server is a single point of failure
- Server costs scale with users
- Server operator controls everything

Peer-to-Peer Architecture



Characteristics:

- Peers connect directly to each other
- No single point of failure
- Infrastructure grows with users
- Users control their own data

How Pear Works

Pear provides three main components that work together:

1. 📊 Data Structures

Store and sync data between peers.

Distributed Logs (Hypercore)

What: Append-only logs that peers can read and replicate
Use for: Event streams, message histories, blockchains
Example: Chat message history

```javascript

const log = new Hypercore('./my-log')
await log.append('Hello, world!')
```

```
const data = await log.get(0) // 'Hello, world!'
```

Key-Value Databases (Hyperbee)

What: Sorted databases built on distributed logs
Use for: Structured data with queries
Example: User profiles, app settings

```javascript

const db = new Hyperbee(core)
await db.put('username', 'alice')
const user = await db.get('username') // 'alice'
```

File Systems (Hyperdrive)

What: Full file systems that sync between peers
Use for: Documents, media, large files
Example: Shared photo library

```javascript

const drive = new Hyperdrive('./my-drive')
await drive.put('/photo.jpg', photoBuffer)
const photo = await drive.get('/photo.jpg')
```

Learn more about Data Structures →

2. 🌐 Networking

Discover and connect to peers automatically.

Peer Discovery (Hyperswarm)

What: Find peers interested in the same topics
How: Uses a Distributed Hash Table (DHT)
Key concept: No IP addresses needed - peers find each other by topic

```javascript

const swarm = new Hyperswarm()
const topic = Buffer.from('my-app-topic')
swarm.join(topic)

swarm.on('connection', (peer) => {
  console.log('Found a peer!')
})
```

DHT (HyperDHT)

What: Decentralized directory service
Purpose: Maps topics to peers
Analogy: Like DNS, but no central authority

Learn more about Networking →

---

3. 🔐 Security

Built-in encryption and authentication.

Public Key Cryptography

Every peer has a **key pair**:

- **Public key**: Shared freely, identifies your peer
- **Secret key**: Never shared, proves you own the public key

End-to-End Encryption

All connections are encrypted by default:

- Peers authenticate each other
- Data is encrypted in transit
- No man-in-the-middle attacks

Content Verification

Data includes cryptographic proofs:

- Verify data hasn't been tampered with
- Verify data came from the right peer
- Works even with untrusted intermediaries

Learn more about Security →

Key Terminology

Topics

A **topic** is a 32-byte identifier that peers use to find each other. Think of it as a "room ID" or "channel".

```javascript
// Generate a random topic
const topic = crypto.randomBytes(32)

// Or use a meaningful string
const topic = Buffer.from('my-app-name', 'utf8')
```

Important: Topics are public! Don't use sensitive data as topics.

Discovery Keys

A **discovery key** is derived from a data structure's public key. It allows peers to find each other without revealing the public key (which grants read access).

```javascript
const core = new Hypercore('./my-core')
const discoveryKey = core.discoveryKey

// Share discoveryKey to let others find and connect
// They still need the public key (core.key) to read data
```

Why it matters: You can announce "I have this data" without revealing what the data is.

Replication

Replication is the process of syncing data between peers. Pear handles this automatically.

```javascript
// Peer A
const coreA = new Hypercore('./peer-a')
```

```
3  await coreA.append('message 1')
4  swarm.on('connection', conn => coreA.replicate(conn))
5
6  // Peer B
7  const coreB = new Hypercore('./peer-b', coreA.key)
8  swarm.on('connection', conn => coreB.replicate(conn))
9
10 // coreB automatically downloads data from coreA
```

Key points:

- Happens in the background
- Efficient (only transfers missing data)
- Verifiable (cryptographic proofs)

Sessions & Snapshots

Sessions let you open multiple views of the same data structure:

javascript

```
1  const core = new Hypercore('./my-core')
2  const session1 = core.session()
3  const session2 core.session()
4
5  // Both share the same underlying storage
6  // Changes in one are reflected in the other
```

Snapshots freeze data at a point in time:

javascript

```
1  const snapshot = core.snapshot()
2  // snapshot.length won't change even if core grows
```

Common Patterns

Pattern 1: Shared Topic (Multi-User)

Use case: Chat rooms, multiplayer games

```javascript
// Everyone joins the same topic
const topic = Buffer.from('game-room-123')
swarm.join(topic)

// All peers connect to each other
swarm.on('connection', (peer) => {
  peer.write('Hi everyone!')
})
```

Pattern 2: Direct Connection (One-to-One)

Use case: Direct messaging, file transfers

```javascript
// Alice generates a keypair
const keyPair = Hypercore.generateKeyPair()

// Alice shares her public key with Bob
// Bob connects directly using Alice's public key
swarm.joinPeer(keyPair.publicKey)
```

Pattern 3: Content-Addressed (Read-Only Distribution)

Use case: Software distribution, content delivery

```javascript
// Publisher creates content
const drive = new Hyperdrive('./content')
await drive.put('/app.js', appCode)

// Users download by public key
const userDrive = new Hyperdrive('./downloads', drive.key)
swarm.join(userDrive.discoveryKey)
// Content syncs automatically
```

Mental Models

Think of Pear as...

📚 Distributed Git

Data structures are like Git repositories that sync between peers automatically.

💬 WhatsApp Without Servers

Messages replicate directly between peers, encrypted end-to-end.

🌊 BitTorrent for Data

But with mutable data, ordered logs, and automatic discovery.

🔗 Blockchain Without Mining

Cryptographic verification without proof-of-work or consensus.

When to Use Pear

✅ Great Fit

- Real-time collaboration tools
- Messaging and chat applications
- File sharing and sync
- Gaming (multiplayer, no game servers)
- IoT device networks
- Local-first applications
- Privacy-focused tools

⚠️ Consider Alternatives

- Public websites (use traditional web hosting)

- Apps requiring powerful server-side computation
- Highly regulated environments requiring audit trails
- Apps where all users must see identical data instantly (strong consistency)

Architecture Decisions

Client vs Server Mode

When joining a swarm, you can be:

```javascript
// Server: Accept incoming connections
swarm.join(topic, { server: true, client: false })

// Client: Actively search for others
swarm.join(topic, { server: false, client: true })

// Both: Maximum connectivity (recommended)
swarm.join(topic, { server: true, client: true })
```

Rule of thumb: Use both modes unless you have a specific reason not to.

Data Structure Choice

Need	Use	Example
Append-only log	Hypercore	Event stream, blockchain
Sorted key-value data	Hyperbee	Database, index
Files and directories	Hyperdrive	File storage, media
Multiple writers	Autobase	Collaborative document

Next Steps

🎯 Build Something

Apply these concepts in a hands-on tutorial
First App Tutorial →

📊 Explore Data Structures

Deep dive into logs, databases, and file systems Data Structures →

🌐 Master Networking

Learn about peer discovery and connections
Networking →

🔐 Understand Security

Cryptography and encryption in Pear
Security →

Questions?

- **Confused about something?** → FAQ
- **Want to discuss?** → Join Discord
- **Found an error?** → Report on GitHub

Edit this page on GitHub

Next page
What is P2P?